Universidad Carlos III de Madrid

Grado de Ingeniería Informática

Proyecto Fin de Carrera

Visión artificial integrada con dispositivos de realidad virtual inmersiva aplicada a videojuegos

Autor: Pablo Sánchez-Herrero Gómez

Tutor: Yago Sáez Achaerandio

Junio 2016

Contenido

Índice ilustraciones

Índice de código

Resumen

El objetivo de este proyecto es crear un simulador de un sable láser. Sin embargo antes de desarrollar esta idea habrá que hacer un estudio previo sobre las diferentes formas de abordar este problema para tratar de buscar el mejor enfoque.

Este documento estará estructurado en los siguientes apartados:

1. **Introducción**: donde se describe la situación actual y por qué es interesante el desarrollo de un proyecto de estas características. También se presentará el problema y que dudas surgirán a la hora de plantearlo.

2. **Estado del arte**: donde se realiza un estudio de los dispositivos de realidad virtual y controladores de movimiento más famosos y relevantes hasta la fecha. El objetivo de este estudio es la obtención de información sobre el funcionamiento de los distintos dispositivos para extraer una posible solución al problema.

3. **Planteamiento del problema**: donde se analizan los datos extraídos del estado del arte y se plantean posible soluciones junto con sus riesgos y alternativas.

4. **Planificación**, **marco regulador** y **entorno socioeconómico**:

   Planificación: donde se explica cómo se va a tratar de descomponer el problema y como se va a repartir el tiempo de desarrollo del proyecto.
   Marco regulador: donde se muestran los distintos aspectos legales que definirán los límites en los que se podrá desarrollar el proyecto.
   Entorno socioeconómico: donde se explicara qué aspectos sociales y económicos actuales afectarán al desarrollo del proyecto. También se valoraran los costes de la creación del proyecto creando un presupuesto del mismo.

5. **Análisis y diseño del problema**: donde se definirán los casos de uso y requisitos que tendrá que implementar la solución final.

6. **Planteamiento de la solución**: donde se explicará cómo se ha ido desarrollando el proyecto, los problemas encontrados y la formas de resolverlos. También se incluirá una sección en la que se muestre una explicación más detallada del funcionamiento de la solución final.

7. **Conclusiones y posibles ampliaciones**: donde se hará un breve resumen de los problemas encontrados más importantes, las mejoras a nivel personal y que es lo que se cree que ofrece la solución final de este proyecto.

   Summary: un resumen en inglés dividido en (1) Introducción, (2) Planteamiento, desarrollo y funcionalidad del proyecto, (3) Conclusiones y posibles ampliaciones.

   Anexo: donde se explicará cómo configurar las librerías utilizadas para el proyecto en el entorno de desarrollo Visual Studio 2015.

Palabras clave: videojuego, OpenNI, OpenCV, Unity, Kinect V1para Windows, Oculus Rift DK1, visión artificial, marcadores, seguimiento.

KeyWords: videogame, OpenNI, OpenCV, Unity, Kinect V1for Windows, Oculus Rift DK1, computer vision, markers, tracking.

1. Introducción

Nos encontramos en un año realmente interesante, tanto para el **sector del videojuego** como para la **visión artificial**, la **realidad virtual** y los **controladores de movimiento**.

El sector del videojuego en la actualidad vive un momento de auge, la existencia de plataformas como Steam o GoG permiten que **la distribución de juegos sea mucho más sencilla**, la presencia de servicios como Steam Greenlight permite también que desarrolladores independientes puedan mostrar sus juegos y posteriormente publicarlos a nivel global. Otro factor importante es la presencia de posible financiación mediante por ejemplo campañas KickStarter permitiendo **que pequeños desarrolladores puedan exponer y poner en desarrollo sus planes e ideas**.

Desde el éxito producido por la consola Wiii en 2006 muchas empresas han intentado aprovechar ese nicho de mercado mediante la creación de diversos controladores de movimiento. El funcionamiento de varios de los dispositivos creados residía en el uso de **técnicas de visión artificial**, mientras que Sony apostaba por la creación del mando Move posicionado por una cámara, Microsoft trataba de convertir el cuerpo del jugador en el propio controlador mediante el uso de la cámara Kinect. Ambas soluciones residían en el uso de técnicas de visión artificial para la captura de movimientos y estás técnicas fueron evolucionando a lo largo de los años, por ejemplo la mecánica del posicionamiento de los mandos Move de Sony está siendo utilizada como mecánica principal para el posicionamiento del dispositivo de realidad virtual el cual saldrá a la venta en octubre de este mismo año.

El mercado de los videojuegos también se está viendo sacudido por la **llegada de varios dispositivos de realidad virtual**, en concreto dentro este mismo año han sido puestos a la venta la versión del consumidor de Oculus Rift (CV1) y el sistema desarrollado por Valve HTC Vive, el cual además cuenta con varios controladores de movimiento de gran precisión. Otras empresas como Sony ya han anunciado la llegada de sus propios dispositivos de realidad virtual al mercado en Octubre de 2016, incluso empresas como Microsoft no dedicadas a la construcción de dispositivos de realidad virtual han mostrado interés por el mercado, presentado una nueva consola para 2017 con las características necesarias para soportar juegos de realidad virtual de forma fluida.

El campo de la visión artificial está también más activo que nunca, **la necesidad de tratar grandes cantidades de datos** como imágenes y videos ha propiciado que muchas empresas se muestren interesadas en este campo. Utilizando técnicas de visión artificial e inteligencia artificial Microsoft por ejemplo ha sacado un programa que trata de reconocer las emociones de las personas de una imagen basándose en sus expresiones fáciles, otra aplicación sacada al público este mismo año es CaptionBot un experimento que trata de describir imágenes subidas por el usuario.

Hay muchos campos interesados en la utilización de visión artificial para la mejora de sus productos, desde aplicaciones para la seguridad como aplicaciones en el campo de la medicina o por ejemplo el marketing de productos y análisis de consumidores. El mercado del automóvil también se ha mostrado muy interesado por la visión artificial, es un hecho que cada vez más empresas están intentando conseguir la creación de un coche autónomo, incluso empresas como Apple la cual ha anunciado la creación de un coche autónomo para el año 2021 aproximadamente. Empresas como Tesla Motors parecen apostar **por el uso de cámaras**

como método principal para garantizar la autonomía de sus coches, el uso de cámaras es usado también por ejemplo por el desarrollador independiente George Hotz el cual está tratando para sacar al mercado un equipo de bajo coste para la transformación de coches estándar a coches autónomos.

Para la asignación de este proyecto mi actual tutor Yago Sáez me propuso realizar **la simulación del movimiento de una espada láser a tiempo real** aplicándolo en un entorno de realidad virtual. Para ello me dejo elegir la mejor forma de abordar el problema.

Tras un estudio previo se decidió hacer uso de la cámara Kinect v1 para Windows junto con el casco de realidad virtual Oculus Rift DK1, ambos proporcionados por el departamento de informática de la Universidad Carlos III de Madrid. La idea era **crear un controlador de movimiento barato** que mediante **técnicas de visión artificial** fuera capaz de transmitir los movimientos del controlador a un juego, intentando garantizar una experiencia lo suficientemente fluida y precisa para ganar el interés del jugador.

Sin embargo el desarrollo de la solución plantea varios problemas, por ejemplo Kinect implementa numerosos algoritmos para la captura de movimientos del cuerpo sin embargo no incluye ningún tipo de funcionalidad implementada que le permita **reconocer y posicionar objetos externos**. Para solucionar este apartado se tendría que recurrir a la creación de un programa que utilice librerías externas para el **procesamiento de la imagen** y la **extracción de la información necesaria**.

Otros problemas que se plantean son: ¿Será capaz el programa de actualizar la posición del controlador **de forma fluida**? ¿**Con que frecuencia** podrá actualizar la posición del sable láser? ¿**Qué limitaciones** plantea el uso de la cámara Kinect? ¿**Qué otras alternativas existen** para la solución del problema? ¿Con que grado de **precisión** podrá representarse el movimiento del controlador físico? ¿Será posible obtener la información necesaria para el posicionamiento y rotación del sable láser **sin la utilización de sensores inerciales**? En caso conseguirse ¿Será posible aplicar los resultados a un juego de forma que pueda generar interés al jugador? ¿Qué mecánicas de juego podrán aplicarse añadiendo el casco de realidad virtual? ¿Qué problemas plantea el uso de realidad virtual y cuáles son sus posibles soluciones?

Para responder a estas preguntas se ha realizado este trabajo de fin de grado, en el que se explorara el funcionamiento de algunos de los dispositivos de control de movimiento más famosos junto con los productos de realidad virtual más relevantes de la actualidad, extrayendo que posibles técnicas utilizadas por estos se podrían aplicar para una solución satisfactoria del problema planteado.

2.Estado del arte

El objetivo de este proyecto es comprobar si es posible realizar un videojuego aplicando realidad virtual y visión artificial **que supla las carencias que puede tener un dispositivo tradicional de entrada de datos**, como por ejemplo el teclado o un mando inalámbrico, permitiendo recrear en un juego el movimiento de una espada láser.

Para tratar de dar con la mejor solución posible se hará un estudio de los diversos dispositivos de entrada, su historia, su funcionamiento, sus ventajas e inconvenientes y sus limitaciones.

Antes de empezar sin embargo se realizará una breve explicación del funcionamiento de los sensores inerciales usados en muchos de estos controladores con el objetivo de que se entiendan mejor las **ventajas y desventajas** del uso de cada uno de ellos.

Sensores inerciales

El **acelerómetro** es un sensor inercial capaz de captar la aceleración lineal. **No es capaz de captar la rotación horizontal pero** sin embargo **puede medir la rotación vertical (inclinación)**. Esto es posible gracias a que conocemos la componente vertical provocada por la gravedad con la cual podemos medir la inclinación del mando. Al solo conocer solo la componente vertical no podemos calcular la orientación lateral.

Poder medir la inclinación permite que muchos dispositvos moviles solo cuenten con un acelerometro para realizar funciones como girar la pantalla o sacar fotos panorámicas. Por ejemplo un movil que solo cuente con un acelerometro podrá detectar si girar la imagen de la pantalla en caso de que el movil rote estando en posición vertical, por el contrario si se apoyara el movil en la mesa y se rotara, la imagen pantalla no giraría ya que no estaría detectando la rotación.

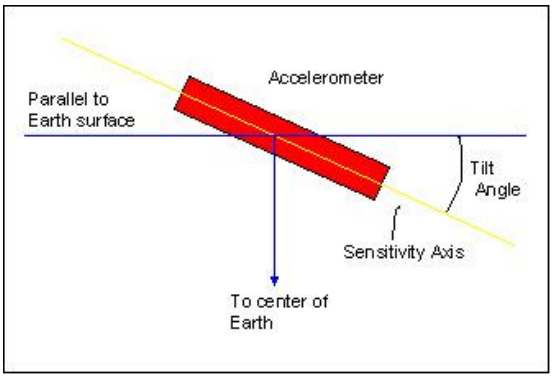


Ilustración 1 Funcionamiento de un acelerómetro para el cálculo de la inclinación

El **giroscopio** se encarga de calcular la **velocidad angular** a partir de un eje, y por lo tanto mostrar los cambios en la rotación del objeto a partir de una orientación inicial [1]pudiendo medir entonces la orientación lateral. El problema del giroscopio es que **irá acumulando error** con el tiempo y deben de ser recalibrados. Los giroscopios utilizados en campos como al aeronautica o ingeniería espacial tendrán un error acumulado mínimo, sin embargo los

utilzados por dispositivos eléctronicos como moviles u similares son mucho más baratos y por lo tanto menos precisos.

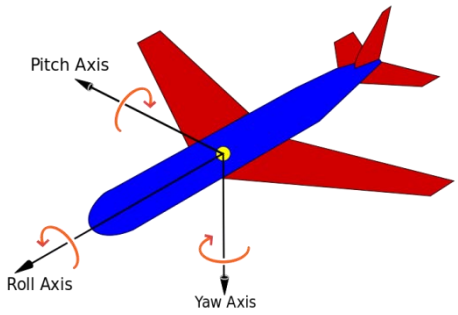


Ilustración 2 Distintos ejes de rotación, los cuales pueden ser medidos a través del uso de giroscopios

Un **magnetómetro**, funciona como una brújula y es capaz de señalar al norte magnético, de manera que mediante este dispositivo al contrario que con los acelerómetros sí que podemos medir la orientación lateral. Sin embargo es bastante susceptible a elementos del entorno que alteren los campos magnéticos.

La unión de estos tres elementos permite reducir el error producido por cada sensor de forma individual, obteniendo la posición y rotación de un objeto de una manera más precisa.

Wiimote(Wii remote)

Wiimote es el dispositivo de entrada principal de la consola Wii. Entre sus componentes se encuentra varios botones, un **acelerómetro de 3 ejes** y un **sensor de rayos infrarrojos** [2]. El acelerómetro le permite captar la aceleración en los 3 ejes de coordenadas X Y Z y está presente también en el "Nunchuk" (controlador adicional al Wiimote que incluye un joystick y varios botones).

Ilustración 3 Wii Mote con Wii Nunchuk

La consola Wii incluye una barra sensora, la cual **contiene cinco LED infrarrojos en cada lateral,** estos LED emiten una luz infrarroja no visible para el ojo humando pero si para el sensor infrarrojo ubicado en el Wiimote. Este sensor permite que tras una configuración inicial se pueda conocer la posición del mando y pueda ser usado como puntero de una forma precisa.

El funcionamiento es el siguiente, los cinco LED infrarrojos estarán ubicados en un **punto fijo**, el sensor del mando por lo tanto los usará como referencia para calcular su posición mediante una triangulación basada en la distancia entre los LEDs.

Ilustración 4 Barra sensora Wii con cinco LED infrarrojos ubicados en ambos laterales.

Una vez el sensor pierde de vista los LEDs es el acelerómetro el que se encarga de calcular la rotación y posición del mando. Estos valores son aproximados y **se irán descompensando a medida que el mando siga moviéndose**. Una vez el sensor detecte de nuevo los LEDs infrarrojos, corregirá su posición y rotación.

Los datos de posición, rotación son enviados vía Bluetooth a la consola que según las especificaciones es capaz de soportar cuatro mandos simultáneamente.

Las posibilidades de este sistema de posicionamiento no solo se limita al entorno de los videojuegos, Johnny Chung Lee demostró en [3] que utilizando un sensor infrarrojo a modo de bolígrafo era capaz de utilizar el mando WiiMote para hacer un seguimiento del LED y crear una pizarra interactiva de bajo coste.

Inicialmente el mando WiiMote iba a contar con un giroscopio, pero para abaratar costes y ya que el acelerómetro podía calcular parte de la rotación y el sistema de triangulación del sensor infrarrojo ofrecía bastante precisión, se decidió no incluirlo.

Más tarde del lanzamiento de la consola se introdujo la extensión para el mando Wii MotionPlus, un accesorio que **incluía un giroscopio** el cual mejoraba la precisión del Wiimote en los casos en los que no podía depender de la triangulación por parte del sensor infrarrojo, **permitiendo movimientos más complejos**. En caso de perder precisión se recomendaba dejarlo unos segundos en una superficie plana, sin embargo también **se podía corregir el error utilizando el posicionamiento por triangulación**. Wii MotionPlus se incorporó como requisito de varios juegos que requerían mayor precisión como por ejemplo Wii Sports Resort.

Ilustración 5 Accesorio Wii Motion Plus

Wii Sports Resort incluía el juego “SwordPlay” el cual hacía uso de los botones del mando y de los acelerómetros y giroscopios para manejar una espada. El juego “The Legend of Zelda: Skyward Sword” hizo uso del mismo sistema de juego de Swordplay.

Ejemplo de mecánica de juego: si presionas b y recibes el ataque de forma perpendicular, bloqueas el ataque.

La diferencia de este juego con otros similares es que el movimiento del mando **se aplicaba directamente al juego** (controlador 1:1). La mayoría de juegos basados en controladores de movimientos se basaban en un sistema de patrones, por ejemplo, si detectas un movimiento lateral aplica el ataque lateral, si el controlador esta aproximadamente arriba y baja de golpe aplica el ataque vertical etc…

Después de un tiempo comenzó a darse a la venta otra versión del mando llamado Wii Remote Plus un mando en el que ya estaba integrado el accesorio Wii Motion Plus.

Ya que el mando hace uso de Bluetooth para comunicarse con la consola Wii es fácil obtener los datos en un ordenador, en [4] muestra una API desarrollada en Unity que recibe la información enviada por el mando y se representa en forma del movimiento de un mando virtual.

PlayStation Move y PlayStation camera

PlayStation Move es un mando creado por Sony Computer Entertainment, fue lanzado a la venta en septiembre de 2010, **mezcla técnicas de visión artificial con varios sensores incerciales** para medir la posición y rotación del objeto (acelerómetro, giroscopio y magnetómetro).

Uno de los aspectos más llamativos del mando es la esfera ubicada en la parte superior del mando, en el interior de esta hay un LED RGB que permite cambiar el color de la esfera.

Ilustración 6 PlayStation Move

Especificaciones PlayStation Move
Plataforma PlayStation 3 (compatible también con PlayStation 4)
Acelerómetro de 3 ejes
Giroscopio de 3 ejes
Magnetómetro
Bluetooth 2.0

PlayStation Move utiliza la cámara PlayStation Eye. Esta camara fue lanzada a la venta bastante antes (octubre 2007), pero no tuvo demasiado éxito debido a la escasa cantidad de juegos que eran compatibles con esta, mas tarde se aprovechó esta camara para crear el sistema de posicionamiento del mando PlayStation Move.

Ilustración 7 PlayStation Eye

Especificaciones cámara PlayStation Eye
Plataforma PlayStation 3
Resolución 640x480 (60 fps)
Resolución 320x240 (120 fps)
Conectividad USB 2.0
Campo de visión 56° o 75°

La camara usa visión artificial para localizar la esfera, esta esfera se iluminará y **podrá cambiar de color** gracias al LED RGB **para diferenciarse del entorno donde se encuentre**. El LED RGB

permite un amplio espectro de colores por lo que aumenta la precisión del dispositivo. El color también cambiara para diferenciarse de otros mandos Move.

Una vez localiza la posición de la esfera procede a calcular su profundidad. Ya que **se conoce de el tamaño de la esfera** puede calcularse la profundidad **midiendo su tamaño actual**, esto permite extraer la localizacion del mando de una forma bastante precisa.

PlayStation Eye tambien permite hacer un seguimiento de la cabeza, aunque este es menos preciso que el de la esfera.

Por otro lado el mando cuenta con una serie de sensores que ayudan a calcular el movimiento y orientación del mando. Cuenta con un **acelerómetro**, un **giroscopio** y un **magnetometro**, la unión de estos tres sensores permite suplir las carencias que tienen por separado y **permiten reducir el error acumulado**. En caso de necesitar corregir el error acumulado PlayStation Move hace uso de la cámara.

Con la unión de la cámara y los sensores el mando es capaz de transmitir **su posición y orientación** en el espacio 3D de una forma bastante precisa, incluso si la cámara no captara la esfera (e.g controlador detrás de la persona) podría seguir funcionando utilizando únicamente los sensores inerciales (cuanto más tiempo este el dispositivo fuera del alcance de la cámara mayor será el error acumulado).

Más adelante junto con el lanzamiento de la consola PlayStation 4 salió al mercado PlayStation Camera, una versión mejorada de la cámara PlayStation Eye. El dispositivo cuenta con **dos cámaras**, gracias a esas dos cámaras puede medir la posición de los objetos y la distancia a la que se encuentran. También incluye otras funciones agregadas, como por ejemplo reconocimiento facial. Al igual que su antecesor PlayStation Eye es capaz de detectar la posición de los mandos PlayStation Move.

Ilustración 8 PlayStation Camera

Especificaciones cámara PlayStation Camera
Plataforma PlayStation 4
1280x800 (60fps)
640x400 pixel (120fps)
320x192 pixel (240fps)
Campo de visión 85°

Kinect

Kinect es un sensor creado por Microsoft (con la colaboración de las empresas Primesense y Rare). En noviembre de 2010 salió al mercado la versión Kinect para Xbox 360. La característica principal que lo diferenciaba de otros dispositivos como el Wii Mote o PlayStation Move era la ausencia de un mando, **siendo el cuerpo del jugador el propio controlador** [5]. También se llegó a plantear que traería una revolución en cuanto a cómo nos comunicaríamos con los ordenadores en el futuro mediante interfaces de tipo NUI (natural user interface) [6].

Ilustración 9 Cámara Kinect V1

En diciembre de 2010 PrimeSense la empresa Israelí detrás de la tecnología 3D de Kinect [7], lanzó al público unos drivers open-source como parte del proyecto OpenNI, junto a este también publicó el middleware NITE [8] [9] [10].En febrero de 2011 Microsoft anunció el lanzamiento de un SDK (software development kit) gratuito no comercial para Windows [11] [12], en junio de 2011 [13] fue publicado .

Más tarde en febrero de 2012 salió al mercado Kinect para Windows a 249$ (150$ para organizaciones educativas), la justificación de la subida de precio respecto a la cámara Kinect 360 (150$) fue que la nueva cámara no les proporcionaría ingresos en ventas de juegos y aplicaciones Kinect [14].

La nueva cámara optimizada para PC permitía el uso del "Near Mode" que permitía ver objetos a 40 centímetros de la cámara con cierto margen de fallo y a 50 centimetros con precisión [15], implementaba mejoras en el seguimiento de esqueleto y mejoras en el reconocimiento de voz [16]. A su vez **permitía el desarrollo de aplicaciones comerciales** mientras que para Kinect 360 creaba un SDK beta [17] con una licencia de fecha límite junio 2016 para desarrollo de aplicaciones no comerciales [14].

La última versión del SDK para Kinect V1 para Windows llegaría en septiembre de 2013 con varios tipos de mejoras, por ejemplo se mejoran las prestaciones de Kinect Fusión cuya funcionalidad permitía usar la cámara como un escáner 3D [18].

Kinect tuvo un indudable éxito, llegando a entrar en el libro Guinness de los récords como el dispositivo electrónico de consumo con las ventas más rápidas vendiendo 133.333 por día durante 60 días [19].

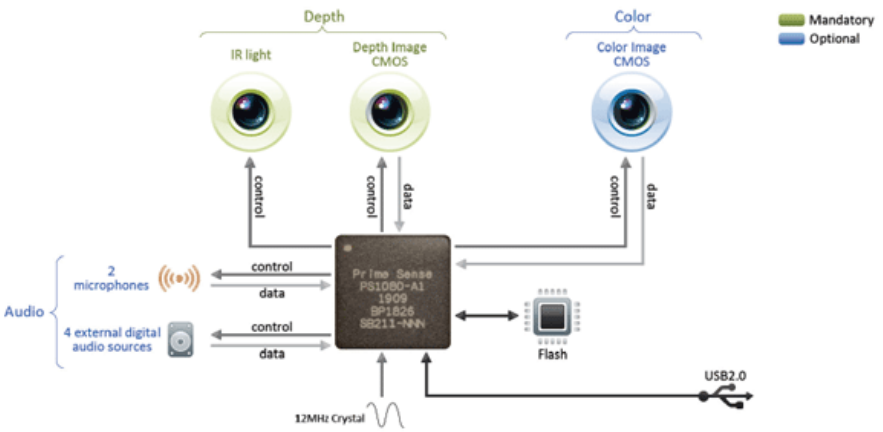


Ilustración 10 Diagrama de la tecnología detrás de la cámara Kinect (Primesense)

Con la llegada de Xbox One al mercado Microsoft **se creó la segunda versión** de la cámara Kinect para Xbox One, esta proporcionaba una mejor precisión, mejorando entre otras cosas la cámara RGB y la forma en la que percibía la profundidad. En julio de 2014 se presentó la cámara para Windows v2 junto con una versión preliminar del SDK 2.0 [20]. En octubre de 2014 se lanzó el SDK 2.0 compatible con la versión v2 de Kinect, permitiendo lanzar las aplicaciones creadas con el SDK a Windows Store junto con otras mejoras que aprovechaban las nuevas funcionalidades de la cámara, a su vez se lanzó a la venta un adaptador que permitía conectar la cámara Kinect Xbox a Windows [21].

En abril de 2015 Microsoft anunció que con el objetivo de crear consistencia entre las diferentes versiones pararían la producción de Kinect para Windows v2 ocupando su lugar Kinect Xbox One con el adaptador anunciado anteriormente. Ya que al contrario que las anteriores versiones (Kinect Xbox 360 y Kinect para Windows v1) **ambas cámaras contaban con la misma funcionalidad y especificaciones** se podría usar con ambas el SDK 2.0 [22].

En la siguiente tabla se muestran las principales diferencias y especificaciones entre las distintas versiones (no se mencionan las diferencias entre el reconocimiento de voz al no ser de utilidad para este trabajo sin embargo en las referencias se pueden explorar las diferencias)

Características	**Kinect Xbox 360**	**Kinect para Windows V1**	**Kinect para Windows V2 y Kinect Xbox One**
Cámara RGB	640x480 (30 fps)	-(RGB, Bayer)**640x480 (30 fps)** -(RGB, Bayer) 1280x960 (12 fps) -(YUV) 640x480 (15 fps)	1920 x1080 (30 fps)
Cámara profundidad	640x480 (30 fps)	**Structured Light** 640x480 (30 fps), 320x240 (30 fps), 80x60 (30 fps)	**Time of Flight** 512x424 (30 fps)
Max. Distancia profundidad	4 m	**Near mode:** 3 m **Default mode:** 4 m	4.5 m (permite hasta 8m pero con posibles errores)

Min. Distancia profundidad	0.8 m	**Near mode:** 0.5 m (sin perdida información) 0.4 m (con posibles pérdidas de información) **Default mode:** 0.8 m	0.5 m
Campo de visión (horizontal)	57	57	70
Campo de visión (vertical)	43	43	60
Motor inclinación cámara	+-27 grados	+-27 grados	No tiene
Juntas de esqueleto reconocidas	20 articulaciones	**Normal mode:** 20 articulaciones (default mode) Mejoras en el seguimiento del esqueleto respecto a Kinect 360 **Near mode:** No asegura las 20 juntas	25 articulaciones Permite el seguimiento de las articulaciones y la detección de gestos de la mano
Max. Seguimiento esqueletos	2	2 (Puede detectar hasta 6 usuarios pero solo seguir el esqueleto de 2)	6
Conexión USB	2.0	2.0 Soporta 4 sensores a la vez conectados en puertos diferentes	3.0 Varias aplicaciones pueden usar el sensor al mismo tiempo

Tabla comparativa de las diferentes especificaciones de los sensores Kinect [15] [23] [24] [25] [26] [27] [28] [29]

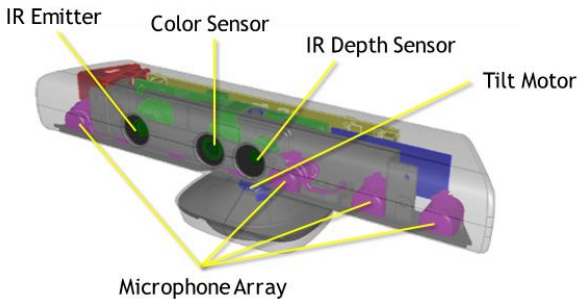


Ilustración 11 Componentes cámara Kinect V1 extraída de [23]

Uno de los componentes de Kinect es la cámara RGB, la cual trabaja a 30 Hz para todas las versiones permitiendo de tal manera obtener **30 FPS como máximo** (frames por segundo) y enviarlos al ordenador o consola vía USB. Para la versión Kinect V1 manteniendo los 30 FPS se obtenían imágenes a color con una resolución de 640x512, la cual es reducida a 640x480 para encajar con la resolución del mapa de profundidad, véase en capítulo 2 RGB Camera de [30]. Con la llegada de la versión Kinect v2 **se aumentó la resolución de la cámara** a 1920x1080 pixeles lo cual supuso una gran mejora en esta.

La primera versión de Kinect contaba con un sensor de profundidad creado por Microsoft y Primesense basado en el principio **Structured Light** (SL). Más adelante con la llegada de Kinect V2 el principio cambió a **Time of Flight** (ToF) apartado 1 [31]. Esta es una de las características que más diferencia una versión de la otra.

El principio Structured Light (SL) se basa en la proyección de una serie de **patrones** sobre un objeto de forma que esos patrones serán deformados al verse proyectados en el objeto. **Analizando la deformación de los patrones** mediante una cámara **se puede calcular la profundidad**. En el caso de Kinect no se usan muchos patrones para permitir una buena tasa de refresco. La cámara Kinect cuenta con un proyector NIR (near infra-red) y una cámara monocroma NIR de la que se obtiene los datos para calcular la profundidad a partir de los patrones, apartado 2 [31].

Mediante el proyector NIR se proyecta una secuencia de puntos predeterminada y se extrae la profundidad de estos a partir de una triangulación entre lo que recibe la cámara NIR y el patrón de puntos original.

Es posible **identificar los puntos** gracias a que cada grupo de puntos infrarrojos tiene una **forma diferente** a los de su alrededor [32] [33].

Ilustración 12 Ejemplo patrones de puntos infrarrojos extraído de [33]

El problema de SL es que **no es muy preciso** ya que necesita **identificar individualmente un punto a partir de sus vecinos**. Para ello necesita agrupaciones de puntos y por lo tanto la superficie de la que queramos calcular la profundidad deberá ser lo suficientemente amplia para poder reconocer un patrón. (e.g no puede calcular la profundidad de un pelo pero si extremidades del cuerpo como brazos) [34].

Para calcular la profundidad mediante ToF hay diversas opciones.

Modulación por impulsos

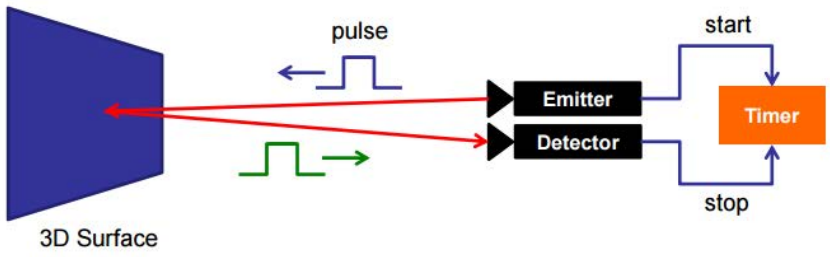


Ilustración 13 Imagen funcionamiento Modulación por impulsos extraída de [35]

Consiste en iluminar mediante pulsos de luz la escena y que la cámara receptora calcule el tiempo que ha tardado en volver el rayo (velocidad luz $3 \cdot 10^8$m/s) [35].

Modulación de onda continúa

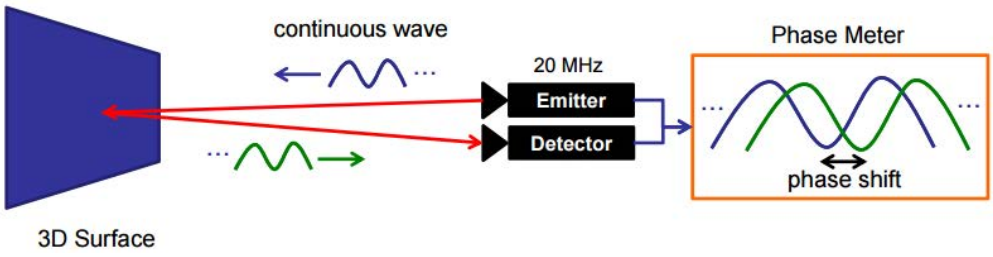


Ilustración 14 Imagen funcionamiento Modulación de onda continua extraída de [13.3]

Kinect utiliza Continuous Wave (CW) Intensity Modulation, apartado 2.2 [31]. El funcionamiento de este principio consiste en comparar el desplazamiento de fase de las ondas que se envían respecto a las que llegan, siendo esta proporcional a la distancia en la que haya sido reflejada [35].

La distancia d se puede calcular de esta manera, siendo c la constante velocidad de la luz y ϕ *phase shift,* $d = c\phi/4\pi$, para más información consultar el apartado 2.2 de [31].

Ambos tipos de cámaras tienen algunos problemas, por ejemplo **la luz ambiental puede provocar pérdidas de información en los mapas de profundidad**, ambas versiones cuentan con un filtro que evita algunos tipos de luz ambiental como la luz infrarroja de los mandos de televisión por ejemplo, capítulo 2 de [30]. Kinect V2 también tiene integrado en el chip un supresor de luz ambiental de fondo, apartado 2.3.1 de [31].

Sin embargo ambos sensores tienen problemas con la luz directa del sol ya que la intensidad de rayos infrarrojos es mucho mayor que la que emite el emisor NIR, por lo cual **Kinect es una cámara no apta para exteriores**, en [36] apartado 5-D por ejemplo prueban la cámara Kinect V1 (SL) en un vehículo en el exterior y la cámara no muestra ninguna información de profundidad.

Otra limitación con la que se encuentra la cámara Kinect V1 (SL) es que la información de profundidad tiene que enviarla a través de una conexión USB 2.0 por lo que necesita reducir la resolución del mapa de profundidad, capítulo 2 de [30], Kinect v2 (ToF) tiene más posibilidades de enviar más información ya que usa el estándar USB 3.0.

El sensor basado en ToF es más preciso que el sensor basado en SL cuya precisión depende de la distancia a la que se encuentren los puntos vecinos. Junto con la mejora de resolución de la cámara RGB, **Kinect V2 mejora bastante su precisión respecto a su predecesor**, especialmente para aplicaciones de escaneo 3D como puede ser Kinect Fusión, sin embargo en las imágenes mostradas a continuación se puede apreciar que la precisión lograda con la cámara Kinect V1 para Windows también puede ofrecer bastante calidad.

Ilustración 15 Imagen de mi escritorio escaneado 3D mediante Kinect fusión SDK 1.8 (Kinect V1)

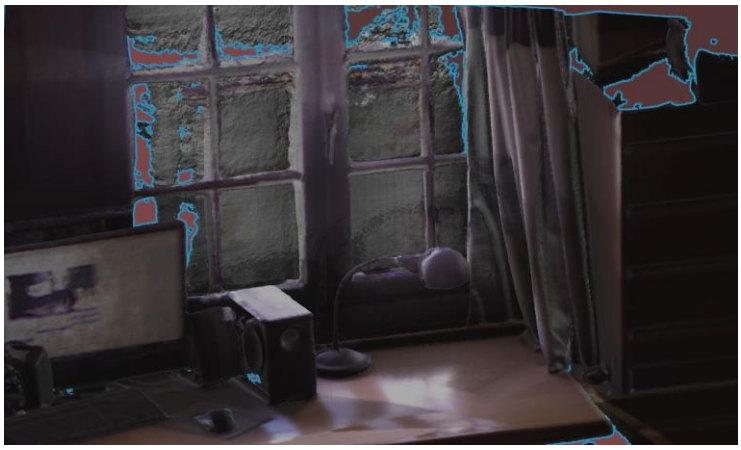


Ilustración 16 Imagen de mi escritorio escaneado 3D con Kinect fusión con color SDK 1.8 (Kinect V1)

Otro problema que se puede plantear es las **interferencias entre varias cámaras Kinect operando simultáneamente**. El problema es más grave en Kinect V1 (SL) ya que al solaparse los puntos no se puede reconocer los patrones, esto es mencionado en [37]. El problema en Kinect V2 (ToF) es más sencillo de solucionar modificando la forma de las ondas, apartado 2.3.2 [31].

Realidad virtual

Realidad virtual (virtual reality VR) es una tecnología por la cual mediante diversos dispositivos se intenta simular de una forma lo más fidedigna posible el mundo real y su interacción con éste.

Dos conceptos muy importantes en los cuales reside la experiencia VR son inmersión y presencia [38].

Inmersión es la sensación que provoca el entorno de realidad virtual que rodea al usuario.

Presencia es la sensación de estar presente en dicho entorno.

Un ejemplo de **inmersión** sería girar la cabeza para mirar un cohete despegando contemplando en el proceso toda la pista de aterrizaje.

Un ejemplo de **presencia** sería intentar saltar físicamente un escalón de un juego para no tropezarse o andar con cuidado en una zona de gran altura.

A lo largo de la historia se han ido inventado varios dispositivos con el objetivo de provocar mayor inmersión, por ejemplo mediante **estereoscopia** (crear ilusión de profundidad proporcionando a cada ojo una imagen diferente). Sin embargo en los últimos años la realidad virtual ha ido obteniendo bastante protagonismo.

Se explicará la historia y el funcionamiento de algunas de las versiones más populares del momento y la comparación de características de unas respecto a las otras.

Oculus Rift

En 2010 Palmer Luckey estudiante de la universidad del Sur de California preparó el primer prototipo de su HMD (Head Mounted Display) en el garaje de su casa y publica sus progresos en un foro de realidad virtual. En dicho foro conoció a John Carmack, cofundador de id Software, el cual estaba trabajando en su propio proyecto de realidad virtual. Una vez vio el progreso de Palmer Luckey preguntó si podría comprar uno de los prototipos, Palmer le envió uno de forma gratuita [39].

En los seis siguientes meses John Carmack trabajó para crear una versión adaptada a VR (virtual reality) del videojuego Doom 3 BFG edition con el prototipo de Palmer [40]. Esta demo fue presentada en el E3 de 2012 **atrayendo la atención de miles de personas**.

En junio de 2012 Palmer Luckey junto con Brendan Iribe, Michael Antonov, Jack McCauley, Nate Mitchell y Andrew Scott Reisse fundan Oculus VR [41].

El 1 de Agosto de 2012 Oculus lanzó una campaña Kickstarter para financiar el desarrollo del HMD (Head Mounted Display) Oculus Rift. En el video promocional de la campaña grandes figuras de la industria del videojuego como Gave Newell o Michael Abrash proporcionaban su apoyo al proyecto lo cual añadió aún más interés por el dispositivo. La campaña finalizo el 1 de septiembre de 2012 con **9.522 patrocinadores aportando 2.437.429 $** [42].

Tras el gran éxito de la campaña empezó la remodelación del casco para poder fabricarlo en masa y en marzo de 2013 empezaron a enviar la versión DK1 (Development Kit 1) a aquellos patrocinadores que hubieran aportado 300$ o más, para más información sobre las modificaciones realizadas respecto al prototipo modificado por John Carmack consultar [43].

Más tarde en agosto de 2013 John Carmack se unió al desarrollo de Oculus como director de tecnología dejando id software en noviembre de ese mismo año [44] [45].

En marzo de 2014 se anuncia la versión DK2 de Oculus Rift con varías mejoras [46] En julio de ese mismo año Facebook compra Oculus VR por 2.000 millones de dólares [47] y empiezan a enviarse la versiones DK2.

En septiembre de 2014 se anunció el prototipo Crescent Bay el cual sería una versión bastante parecida a la versión final destinada al consumidor, entre algunas de sus mejoras

proporcionaba 360° de seguimiento del casco gracias a la ubicación de varios LEDs infrarrojos en la parte trasera [48].

El 6 de enero 2016 se abrió la pre compra de la versión final Oculus Rift CV1 (consumer version) [49], también se anunció que todos los patrocinadores de KickStarter que hubieran superado los 300$ recibirían dicha versión de forma gratuita, véase informe en [42] . La fecha de venta oficial sería en marzo, pero debido a problemas de escasez de piezas se anunció retrasos en las entregas de los pedidos estimando fechas de entregas de junio en adelante [50].

DK1 (development kit 1)

La primera versión para desarrollo sacada al mercado gracias a la campaña KickStarter cuenta con

- Un casco conectado a una caja de control, la cual permite modificar parámetros como el contraste por ejemplo.
- Un cable USB para conectar la caja de control con el PC
- Cables DVI y HDMI para conectar la caja de control con el PC
- Tres pares de lentes, adecuadas para varios tipos de vista.

Ilustración 17 Oculus Rift DK1

El casco cuenta con varias tiras para ajustar el dispositivo a la cabeza además de una rueda para ajustar la profundidad de la pantalla.

Incorpora una **única pantalla LCD** con resolución de 1280x800 (características extraídas de [38]) la cual muestra imágenes con una **tasa de refresco de 60 Hz**. Además cuenta con un conjunto de sensores inerciales (acelerómetro, giroscopio y magnetómetro) que le ayudan a calcular la orientación del casco.

Las lentes separan en 2 las imágenes de la pantalla (resolución 640x800 por ojo). El problema de tener la pantalla tan cerca provoca el efecto “screen door” en el que se puede ver el borde que separa los pixeles de la pantalla.

El efecto de inmersión se consigue

- **Mostrando un campo de visión amplio** (aproximadamente 90 grados verticalmente y 110 horizontalmente) gracias a la pantalla y a las lentes.

- **Mediante estereoscopia** (mostrando una imagen en cada ojo) **provocando el efecto de profundidad**. Para ello la aplicación debe de haber sido construida para trabajar

con realidad virtual, cada ojo recibirá información de la escena desde la posición donde se debería encontrar el ojo.

- Respondiendo acorde con los movimientos del casco, **utilizando los sensores inerciales** Oculus detecta la orientación de la cabeza del jugador, representando el entorno función de ello, este tipo de libertad es comúnmente llamado 6DOF (six degrees of freedom).

Hacer que la experiencia sea lo más fluida posible **es un proceso costoso**, ya que el propio casco introduce información que tiene que ser representada a tiempo real y es necesario renderizar y distorsionar 2 imágenes al mismo tiempo (1 por ojo) [38].

Debido a que solo cuenta con los sensores inerciales para el captar el movimiento del casco **no es posible calcular la posición del casco** con la suficiente precisión, de modo que únicamente es utilizada la orientación de este.

DK2 (depelopment kit 2)

La segunda versión puesta a la venta cuenta con:

- Un casco de realidad virtual
- Una cámara infrarroja
- Dos pares de lentes
- Cables y adaptadores

Ilustración 18 Oculus Rift DK2

Cuenta con varias correas para ajustar el casco a la cabeza del usuario junto con una rueda para ajustar la distancia de la pantalla.

Cuenta con una **única pantalla OLED** de resolución 1920 x 1080 (960x1080 por ojo) con una tasa de refresco de **75 Hz**, una IMU (inertial measurement unit, compuesta de un acelerómetro, giroscopio y magnetómetro) que al igual que su predecesor ayuda a calcular la orientación del casco (la IMU opera a 1000Hz pero es necesario el sistema de seguimiento "Constellation" para corregir el error acumulado). Para el calcular el posicionamiento del casco este tiene una serie de **marcadores infrarrojos** distribuidos por su cubierta los cuales son localizados por una cámara infrarroja. La cámara cuenta con un espejo en su lente que filtra la luz que no sea infrarroja, el sistema de posicionamiento es llamado "Constellation" y puede realizar tracking de la parte frontal del casco pero no de la trasera (en la parte frontal están los marcadores infrarrojos).

Para más información sobre las características de las versiones DK1 y DK2 consultar capítulo 1 de [38].

DK2 utiliza una pantalla **OLED** (organic light emitting diode) es decir cada diodo puede emitir luz individualmente lo cual permite obtener negros "verdaderos", el uso de OLED permite el uso de la técnica "low persistence", lo cual **evita problemas como el mareo**.

¿Por qué es producido el mareo?

Si el usuario está mirando un pixel, al rotar la cabeza la pantalla deberá actualizarse para actualizar la posición del valor de ese pixel, el problema es el espacio de tiempo que hay entre esa actualización. Para que el ojo humano no percibiera que el pixel aún no se ha actualizado se estima que la pantalla debería tener una tasa de refresco de 1000 Hz [51] (bastante alejado de las pantallas usadas en la actualidad).

El ojo por lo tanto esperará que el pixel se mueva y seguirá el trayecto que debería hacer, **al no notar esa actualización el ojo volverá a fijarse en el pixel** el cual no ha modificado su posición, esto provocará el mareo del usuario (cuanta más tasa de refresco menos mareo).

Sin embargo es posible utilizar la técnica "**low persistence**" para evitar el problema. Consiste en dejar la **pantalla en negro entre los espacios de actualización** evitando las inconsistencias percibidas por el ojo, si esto se realiza rápidamente el ojo no percibirá los espacios en negro. En el video [52] a partir del minuto 6 se puede ver una buena demostración del problema, en [51] Michael Abrash (antiguo empleado de Valve, actual jefe científico de Oculus) explica más en detalle el problema. En [53] hay un ejemplo bastante interesante que muestra el efecto que produce.

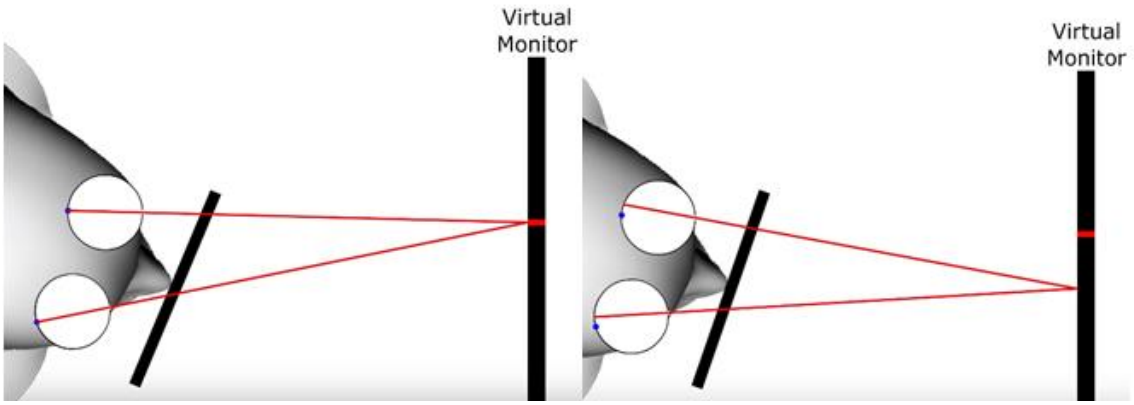


Ilustración 19 Ejemplo del efecto producido sin "low persistence" extraído de [52]

Consumer versión (CV1)

Esta es la versión final del producto, incluye varias mejoras, entre ellas:

- Mayor resolución, esta vez son **2 pantallas OLED** con resolución 1080x1200
- **90 Hz** de tasa de refresco
- **360 grados de posicionamiento** del casco mediante el sistema "Constellation" (incluye marcadores infrarrojos en la parte trasera del casco)

Ilustración 20 Oculus Rift CV1

Aunque aún no haya sido anunciada la fecha de salida en las fechas en las que se publica el proyecto, Oculus cuenta también con los mandos Oculus Touch los cuales utilizan un sistema de seguimiento parecido al del casco. Cuentan con varios marcadores infrarrojos que son posicionados por la cámara infrarroja (“Constellation”). Para calcular la rotación se hace uso de las IMUs [54].

Ilustración 21 Oculus Touch

PlayStationVR

La apuesta de Sony por el campo de realidad virtual se dio a conocer en el evento GDC (game developer conference) de marzo del 2014 [55], fue presentado como “Proyect Morpheus" pero más tarde en 2015 se cambió su nombre a PlayStation VR. Aunque fue presentado a principios de 2014, el proyecto había sido comenzado tras finalizar el desarrollo de los mandos PlayStation Move en 2010 [56], el sistema de posicionamiento de los mandos fue utilizado para desarrollar varios prototipos.

Ilustración 22 Prototipos “Project Morpheus”

El 8 de octubre de 2015 fue comprada la empresa belga SoftKinetic desarrolladores de cámaras con tecnología ToF (tecnología usada por Kinect en la versión v2) [57]. En el GDC de marzo de 2016 se anunció el precio, fecha de salida y características del dispositivo. PlayStation no aceptará juegos para su dispositivo que se mantengan en una tasa de refresco estable superior a los 60 FPS. Una vez alcanzado esta cota es posible aumentar la tasa de refresco hasta 120 FPS utilizando la tecnología de Sony para reduplicar algunos frames [58]. El dispositivo contaría con una tasa de refresco 120Hz (según Oculus la mínima tasa de refresco debería ser 90 Hz [59]).

Ilustración 23 PlayStation VR

El casco posee 9 LEDs que permitirán posicionar el casco en el espacio 3D, para ello se hace uso del dispositivo PlayStation Camera. El sistema de posicionamiento es similar al usado por los mandos PlayStationMove. El seguimiento de los LEDs se realizara 1000 veces por segundo según las especificaciones. Además del seguimiento del casco la cámara también servirá para ubicar los mandos Move y mandos dualshock 4.

HTC Vive

Gran parte de la información sobre la historia del dispositivo ha sido extraída del evento GDC (Game Developers Conference) 2015 en la que se realizó un pequeño museo con prototipos e información del proceso seguido para la creación del dispositivo HTC vive [60].

Ilustración 24 HTC Vive

En Mayo 2012 Valve **empezó apostando por técnicas de visión artificial** usando varios marcadores fiduciarios AprilTag (son unos marcadores QR desarrollados por Ed Olson utilizados como puntos de referencia [61] [62]) y una cámara que los detectará. Cubriendo una habitación con varios marcadores eran capaces de posicionar una cámara.

En enero de 2013 desarrollaron otra cámara prototipo la cual incorporaba una única pantalla y una IMU (inertial measurement unit, dispositivo que informa de la aceleración (acelerómetro), velocidad angular (giroscopio) y campos magnéticos (magnetómetro)). Mediante el uso de marcadores fiduciarios y la IMU era capaz de posicionar la cámara. A su vez hicieron experimentos con pantallas **AMOLED**, permiten controlar la luminosidad de cada pixel y hacer uso de la técnica "**low persistence**", técnica que acabo siendo adoptada por los dispositivos Oculus Rift. En marzo de 2013 Valve añadió un modo VR para el videojuego Team Fortress 2 [63].

En abril de 2013 desarrollaron un casco utilizando algunos de los elementos anteriores, contaba con 2 pantallas individuales AMOLED y la cámara. **Mediante los marcadores fiduciarios se conseguía** de nuevo **posicionar el casco**. Este prototipo fue creado para que posibles colaboradores entendieran el potencial de la realidad virtual. Más tarde en septiembre se desarrolló una demo conocida como "The Room" con el objetivo de realizar pruebas con el HMD (head mounted display).

Ilustración 25 Imagen de la habitación de desarrollo de "The Room" extraída de [60]

En septiembre de 2013 se desarrolló el **primer prototipo de seguimiento mediante infrarrojos**, el cual es similar a la tecnología usada para la versión del consumidor de 2016. Este prototipo surgió ante la necesidad de crear un método de seguimiento que no implicara

llenar una habitación de marcadores. Para ello usaban entre otros elementos infrarrojos y unos discos duros modificados.

En octubre de 2013 se creó otro prototipo con el objetivo de también **evitar posicionar marcadores en la habitación**, para ello la cámara pasaba a estar fuera del casco y el casco estaba recubierto de varios puntos (marcadores). Mediante la cámara se detectaban los marcadores fiduciarios del casco obteniéndose la posición y orientación de este.

Ilustración 26 Prototipo de HMD y cámara con puntos como marcadores extraído de [60]

En enero de 2014 se presenta al público SteamVR SDK y el modo VR en Steam, además se realizó una conferencia presentando las instalaciones y la demo “The Room”.

En mayo de 2014 se crea el **primer HMD** (head mounted display) **basado en la tecnología de seguimiento láser** desarrollada en septiembre del año anterior. En octubre de ese mismo año se hacen pruebas para hacer el seguimiento de un mando con la misma tecnología.

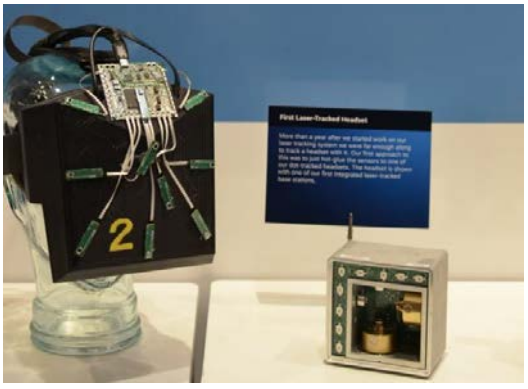


Ilustración 27 Prototipo HMD basado en tecnología de seguimiento láser extraído de [60]

En noviembre de 2014 se crea otro HMD (V minus 1) basado en el seguimiento mediante láser, se hacen por primera vez varias réplicas siendo usadas por colaboradores como HTC para desarrollo. En diciembre crean un prototipo para los controladores de movimiento.

En febrero de 2015 se mejora la estación láser y en el GDC 2015 de marzo se presentan el HMD, mandos y sistema de seguimiento al público.

El lanzamiento oficial de HTC vive es realizado el 5 de abril de 2016.

¿Cómo funciona el sistema de seguimiento del HMD (head mounted display) HTC vive y de sus mandos?

Valve hace uso del sistema **lighthouse** para hacer el seguimiento de su HMD y de sus controladores, para ello utiliza dos estaciones (lighthouses) que utilizan luz infrarroja para que

los controladores y el HMD puedan calcular su propia posición. Una estación o caja cuenta únicamente como entrada una toma de corriente, es decir **ni recibe ni envía información al ordenador o a los controladores** como otros tipos de dispositivo (e.g sistema de seguimiento que utiliza Oculus, "Constellation").

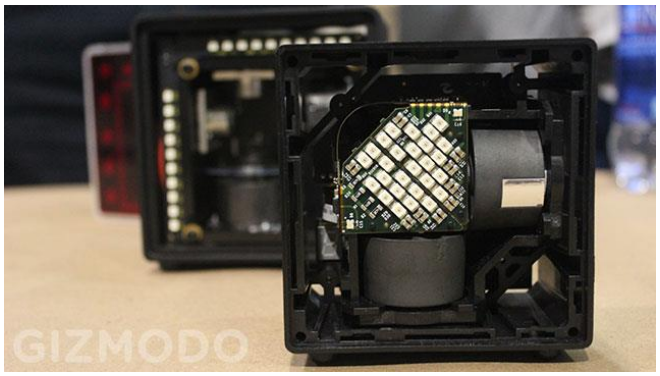


Ilustración 28 Imagen de estaciones lighthouse extraído de [9.2]

Cada estación cuenta con un **LED infrarrojo estático**, y con **dos emisores de luz infrarroja que rotan**, al rotar inundan la habitación con luz infrarroja (como si estuvieran escaneando la habitación), uno de forma vertical y otro horizontal. Los mandos y el HMD están equipados con unos **fotosensores** que detectan la luz infrarroja, a su vez estos están conectados a un **temporizador**.

Cuando el LED infrarrojo estático de la estación se active mandara una señal de luz infrarroja en todas las direcciones. Cuando los fotosensores del mando y HMD detecten la luz activaran el temporizador, tras el "flash" de luz del LED estático se activará uno de los rotores de luz infrarroja proyectando una línea infrarroja, una vez termine de rotar volverá a producirse un flash tras este se activará la rotación del segundo (hay un rotor que traza la línea de forma horizontal y otro que la traza de forma vertical). En el momento que la luz pase por el fotosensor anotará el tiempo tanto para el rotor horizontal como para el vertical. Una vez terminen de ejecutarse las dos rotaciones se volverá a repetir el ciclo de nuevo (flash, activación un rotor, flash activación otro rotor).

Ilustración 29 Representación del funcionamiento del sistema lighthouse extraído de [64]

Ya que **se conocen** la **velocidad de los rotores** y la **distribución de los fotosensores**, es posible **calcular a partir del tiempo** de llegada de la luz de los barridos del rotor horizontal y vertical a cada fotosensor **la posición del dispositivo** (tanto del casco como de los controladores).

En el video [64] se ve de forma muy sencilla el proceso que sigue el sistema.

Para poder hacer el seguimiento en **360 grados** se cuenta con dos estaciones lighthouse las cuales se pueden sincronizar entre si mientras tengan a la vista a la otra (en caso de que la línea de visión entre ambas no sea buena también es posible usar un cable). Los rotores giran 60 veces por segundo, sin embargo esto puede no ser suficiente en algunos casos. Tanto el casco como los controladores cuentan con una IMU que permite calcular la orientación de los objetos de una forma más rápida (1000Hz) sin embargo acumula error. Como solución **se utiliza la información obtenida mediante lighthouse para corregir el error acumulado** de los sensores inerciales [65].

Un empleado de Valve anónimo comento que el sistema era además muy **barato de producir** [66], lo cual supone una ventaja considerable.

El HMD cuanta también con una cámara frontal que permite al usuario activar el modo "Chaperone", el cual permite visualizar el entorno sin necesidad de quitarse el casco.

Comparación

	Oculus Rift CV1	**PlayStation VR**	**HTC Vive**
Pantalla y resolución	2 x OLED 1080x1200(total 2160x1200)	1 x OLED 1920x1200	2 x OLED 1080x1200 (total 2160x1200)
Tasa de refresco	90 Hz	120 Hz	90 Hz
FOV horizontal	110°	100°	110°
Seguimiento de casco	360° Posicionamiento mediante "Constellation Tracking System"	360° Hace uso de PlayStation Camera para seguimiento de LEDs	360° Y posicionamiento en el rango de las estaciones lighthouse
Sensores	IMU Cámara infrarroja	IMUs	IMU 2 Estaciones lighthouse
Otros			Cámara frontal, modo "Chaperone"
Controladores	Mando Xbox Oculus Touch (sin fecha de salida)	Dualshock 4 PlayStation Move	Mando Xbox HTC vive controllers
Fecha de salida	Marzo 2016	Octubre 2016	Abril 2016
Precio de salida	€699, $599, £500	44,980 yen, $399 USD, €399 £349	€899, $799, £689

3.Planteamiento del problema y alternativas de diseño

El cliente, en este caso mi tutor de fin de grado Yago, me propuso crear una **simulación de un sable láser** dándome a elegir como quería abordar el problema.

La primera fase, antes de la propia asignación del proyecto fue crear una serie de requisitos mínimos para la simulación o juego. Tras la creación de los requisitos mínimos se procedió a realizar una rápida búsqueda para tratar de informarme sobre proyectos similares y con qué dispositivos era posible realizar la tarea.

La primera idea que me proporcionó Yago fue la de usar el móvil como dispositivo de entrada utilizando los **sensores del móvil**. Antes de estudiar esa opción a fondo preferí estudiar el funcionamiento del mando de la consola Wii ya que este también contaba con sensores similares. Haciendo un estudio preliminar concluí que **el mando de Wii no era adecuado para la resolución del problema**, aunque dentro del apartado del estado del arte se explica más detalladamente el funcionamiento de este dispositivo comentare las causas de tal decisión.

El mando Wii original cuenta con un acelerómetro, este acelerómetro le permite calcular la aceleración producida en los tres ejes además de la inclinación del mando (ya que se conoce la componente vertical (gravedad)). Sin embargo **el acelerómetro no percibe la rotación horizontal** (yaw) sobre sí mismo, además con el tiempo el **acelerómetro va incrementando su error acumulado**. Para solucionar esto la consola Wii incluye una barra sensora, la cual posee 10 LEDs infrarrojos (5 en cada extremo), invisibles para el ojo humando pero detectables por el sensor infrarrojo ubicado en el extremo del mando.

La barra es situada en un punto fijo y sirve como referencia al mando de forma **que mediante triangulación puede detectar la posición y rotación del mando** siembre que el sensor este orientado hacia la barra. En caso de que los LEDs de la barra no sean visibles por el sensor infrarrojo el mando **recurrirá al acelerómetro** para calcular el movimiento del mando y cuando el sensor infrarrojo del mando detecte de nuevo los LEDs se corregirá el error acumulado por el acelerómetro. Este sistema permite que por **breves periodos de tiempo** el sensor no necesite la visibilidad de la barra.

El problema de la **degradación de precisión por falta de visibilidad** de la barra sensora fue paliado con la inclusión de la expansión Wii Motion Plus, la cual **incluía** en el mando **un giroscopio de tres ejes**. De esta manera el mando podía calcular la rotación sin necesidad del sensor de LEDs. Sin embargo a pesar de que mejorara de forma notable la precisión seguía necesitando de vez en cuando corregir el error acumulado mediante el sensor infrarrojo.

Tras ese estudio preliminar y revisando los requisitos mínimos me decante por **descartar el mando de Wii como dispositivo de entrada**. Era posible enviar la información al ordenador ya que usaba Bluetooth sin embargo el jugador debería de vez en cuando orientar el mando hacia el sensor para calibrarlo lo cual **era un inconveniente en la inmersión del jugador** dentro del juego, además no se disponía del mando WiiRemote ni de la expansión Wii Motion Plus.

Tras descartar el mando de Wii también **se descartó el uso de un teléfono móvil** como dispositivo de entrada debido a las siguientes razones: los sensores inerciales eran iguales o peores que los disponibles en el Wii Remote (en concreto mi teléfono móvil LG l5 solo disponía

de un acelerómetro). Más tarde durante el desarrollo del proyecto se pensó una manera de mejorar la precisión del móvil el cual será explicado en las conclusiones del proyecto.

Otros dispositivos estudiados fueron PlayStation Eye, PlayStation Move y Play Station Camera. Tanto PlayStation Eye como PlayStation Camera percibían la esfera ubicada en la parte superior del mando PlayStation Move **mediante visión artificial**, de esta manera podía localizar la posición del mando en dos dimensiones. **Ya que se conocía** de antemano **el tamaño de la esfera**, **era posible calcular la profundidad** a la que se encontraba en función su tamaño actual, pudiendo ubicar de esta manera el objeto en tres dimensiones. Para calcular la orientación se ayudaba de un acelerómetro, un giroscopio y un magnetómetro.

Debido a que no se contaba con ninguno de dichos dispositivos no se dedicó mucho tiempo a valorar en profundidad esta opción, sin embargo la comprensión de su funcionamiento **ayudo a orientar la idea del proyecto final**.

Aunque no se estudiaron con detalle debido a su no disponibilidad se buscó información de otros dispositivos como Razer Hydra, Oculus Touch, y los controladores HTC Vive.

Buscando proyectos similares se encontró el realizado por Benjamin Teitler [67] [68], el cual **mediante visión artificial**, Oculus rift (DK1) y el mando PlayStation Move **podía moverse e interactuar en un entorno de realidad virtual de una forma muy precisa**. El principal problema era que para hacer el seguimiento de la cabeza y del mando utilizaba 16 cámaras Optitrack Flex 13 [69], cada una operando a 120 fps con una resolución de 1280x1024 cada una valorada en 999 $, en total estimaba que el proyecto costaba alrededor de 20.000$. Aunque el coste era bastante elevado hizo que surgiera la idea de intentar usar **visión artificial con marcadores** para hacer el seguimiento del sable láser.

Más tarde encontré un proyecto en el que mediante la librería de visión artificial OpenCV era capaz de localizar unos marcadores con diferente color y de extraer sus coordenadas [70]. A raíz del video busque información sobre cómo hacer un seguimiento de colores con el SDK de Kinect pero me descubrí que **Kinect estaba orientado al seguimiento de partes de cuerpo**, no al de objetos externos por lo que en principio **tendría que recurrir al uso de una librería externa** para realizar el seguimiento de los marcadores.

El departamento de informática de la Universidad Carlos III además de prestarme Kinect V1 para Windows me prestó el casco de realidad virtual que tenía disponible, era la versión DK1 de Oculus y me prestaba la funcionalidad deseada para el proyecto.

Una vez hecho en análisis previo del problema ya en febrero se procedió a una búsqueda más profunda sobre cómo afrontar el proyecto, en esta fase se buscó enriquecer la información realizada en la primera estimación del proyecto (antes de febrero), se busca información acerca del funcionamiento de Kinect y las alternativas para tratar sus datos, se busca información sobre proyectos similares y bibliografía que pueda ayudar a realizar el proyecto. Tras hacer esa recopilación se realiza una evaluación de riesgos de los que sufre proyecto.

En caso de necesitar la librería externa OPENCV la opción más sencilla sería usar C++ usando el IDE (Integrated Development Environment) Visual Studio para la implementación de la solución.

Evaluación de riesgos inicial

- Sería la primera vez en utilizar tanto el lenguaje como el IDE.

-Primera vez usando un dispositivo Kinect o un casco de realidad virtual

-Primera vez intentando solucionar un problema de visión artificial, **sin conocimientos previos de la materia**.

-**Posibles actualizaciones de versiones** de software a lo largo del proyecto que afecten a la implementación, especialmente problemático para Unity.

-**Actualizaciones de información** respecto a la realidad virtual **a medida que transcurre el periodo del proyecto**, aun no se sabe toda la información ya que muchos de los dispositivos aún no han salido al mercado.

-**No se está seguro si es posible llegar a una solución viable**, limitado a las características técnicas de la cámara Kinect, por ejemplo, con la cámara se pueden obtener un **máximo de 30 imágenes por segundo**, no se sabe el tiempo que puede tardar en procesarse una imagen ¿se podrá por lo tanto obtener una tasa de actualización del sable láser lo suficientemente fluida?).

Sobre el desarrollo en Unity tenía algún conocimiento previo, lo cual me facilitaría la tarea final de implementar un juego para demostrar las capacidades del sable láser. Se ha elegido Unity frente a CryEngine o UDK por ser mucho más sencillo y adecuado a las características del proyecto.

Evaluando el tipo de proyecto y los riesgos que conllevaba **se decidió optar por el enfoque de técnicas agiles** de desarrollo, en concreto un ciclo de vida basado en el **prototipado desechable** y el **prototipado evolutivo**. Más información de dichas técnicas en Pag 362 de [71]

¿Qué **ventajas** conlleva escoger este enfoque en vez de uno más clásico?

- Permite la construcción de un sistema cuyos **requisitos no se conocen de una forma clara o concisa**.
- La utilización de un desarrollo iterativo e incremental **permite reaccionar ante posibles cambios**.
- Es posible enviar resultados al cliente (tutor) en la etapa de desarrollo
- Permite una mejor **comprensión del problema** antes de la implementación final
- Se **reduce el riesgo y la incertidumbre**, el desarrollo de un prototipo funcional **permite demostrar la viabilidad del proyecto**, véase página 374 de [71]

¿Porque **una mezcla** de **prototipado desechable** y **prototipado evolutivo**?

Ya que el problema es nuevo para el desarrollador y el **riesgo de incertidumbre es alto**, especialmente en las primeras etapas del desarrollo, la utilización de un **prototipado desechable** permite **orientar el trabajo de una forma rápida**. Además ya que muchas funcionalidades serán descartadas se evitara la creación de documentación que posiblemente sea descartada. Normalmente el prototipado desechable es usado utilizando papel u otras herramientas que faciliten la creación de un prototipo inicial rápido. En el caso del proyecto consideré que utilizar únicamente ese tipo de prototipos no ayudaría tanto a orientar el proyecto como unos prototipos funcionales.

Una vez el proyecto este **orientado** y **la incertidumbre se vea reducida**, es de interés que en las siguientes etapas de desarrollo se genere un prototipo más robusto y que vaya siendo refinado.

Iterar permite que se puedan agregar requisitos que en un principio no podrían haber sido concebidos. Con el suficiente refinamiento el prototipo incluso puede llegar a convertirse en el producto final.

Tras la búsqueda de información inicial y elegir el enfoque que debería llevar el proyecto se realizó una primera educción de unos requisitos generales. En la elaboración de dichos requisitos se decidió que el **desarrollo de la solución** del proyecto quedaría **separado en dos partes**.

1. **Programa de seguimiento**: encargado de realizar todas las tareas de visión artificial como el reconocimiento y obtención de las coordenadas 3D de los marcadores.
2. **Juego**: encargado de recibir los datos proporcionados por el programa de seguimiento y aplicarlos en un pequeño juego que permita el uso de un casco de realidad virtual.

La separación del software en dos partes planteaba el problema de cómo comunicar ambos, sin embargo conllevaba muchas ventajas

- Es posible implementar una sin tener que depender de la otra, **en caso de no poder avanzar más con una se puede continuar con la otra**.
- **Se reduce el riesgo** de que modificar una parte conlleve la posibilidad de introducir un error en la otra.
- Permite comenzar la fase de creación del juego a una etapa más tardía y dedicar la mayoría de recursos al programa de seguimiento el cual es el que más incertidumbre y riesgo genera. El poder reducir la etapa de desarrollo del juego también **reduce la posibilidad de modificación del proyecto debido a actualizaciones** de Unity o del SDK de Oculus Rift.
- Permite organizar mejor la planificación y el desarrollo del proyecto.
- Facilidad a la hora de realizar las baterías de pruebas.
- **Facilidad a la hora de crear posibles ampliaciones o mejoras** del proyecto. Si queremos mejorar por ejemplo los gráficos del juego no se necesitaría acceder al programa de seguimiento.

Requisitos iniciales

- Se debe tratar de **aumentar la frecuencia de actualización** de posicionamiento del controlador **al máximo posible** con un mínimo de 15 y 30 el máximo posible de actualizaciones por segundo (debido a las especificaciones técnicas de la cámara Kinect V1 para Windows (30FPS)).

- El proyecto constará de **dos partes independientes**, una dedicada al **seguimiento y extracción de coordenadas** de los marcadores y otro que **procese la información** a partir de los datos del primero aplicándolos en un **juego**.

4.Planificación, marco regulador y entorno socioeconómico

Planificación

Como se ha comentado en la sección anterior se hará uso de **técnicas agiles** como **prototipado evolutivo** o **prototipado desechable**, por lo tanto **las fases de especificación de requisitos y diseño irán ligadas al desarrollo** iterativo de prototipos, no a una única fase.

El proyecto contará con **dos partes** a desarrollar, **el programa de seguimiento** y **el juego** que aplique la información obtenida por el primero.

El inicio de proyecto será el 1 de febrero de 2016 terminando el 22 de junio del mismo año siendo un total de 143 días.

Para la estimación de horas a dedicar se ha utilizado la normativa de estudiantes para el trabajo de fin de grado la cual especifica un trabajo de **300 horas** [72] por parte del alumno, lo cual deja como media un trabajo de **2 horas y 6 minutos diarios**. Para garantizar su cumplimiento mínimo se hará un seguimiento de las horas dedicadas al proyecto durante mínimo un mes.

Debido a los **riesgos** especificados en la anterior sección la planificación inicial deberá ser bastante **flexible**, ya que se cuenta con muchos elementos desconocidos y nuevos a desarrollar.

En la fase previa al inicio de proyecto antes de la asignación del TFG hay un periodo de 10 días que no se tiene en cuenta en la planificación pero que sirve como análisis preliminar al proyecto.

Se planteó la **creación de videos** de funcionamiento de los distintos programas para **poder explicar** al tutor de una forma más clara el **estado de desarrollo** y **viabilidad del proyecto**. En la realización de esos videos se mostrarán las distintas funcionalidades del proyecto implementadas por lo que servirán también como método de pruebas para **verificar la consecución de objetivos**.

La mayoría de tiempo de desarrollo será asignada al programa de seguimiento debido a la falta de conocimientos previos y los demás riesgos mencionados en el apartado del planteamiento del problema.

Tras los informes al tutor se asignará una franja de tiempo para aplicar el feedback del tutor.

Dada la naturaleza de TFG no se ha considerado una fase de mantenimiento.

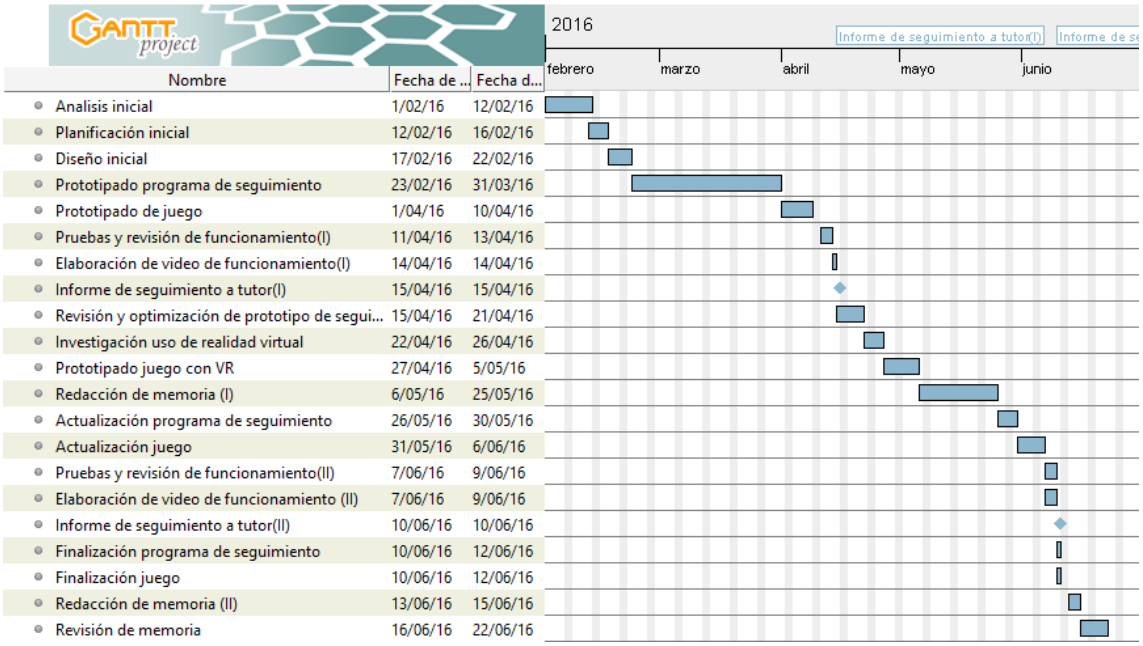


Ilustración 30 Diagrama Gantt Planificación

Marco regulador

Para el desarrollo del proyecto habrá que revisar posibles **restricciones de tipo legal**.

Marco regulador de programa de seguimiento

Para el desarrollo del programa de seguimiento se hará uso del IDE **Microsoft Visual Studio Community 2015** el cual permite el **uso gratuito** de su software para aplicaciones no comerciales y para aplicaciones comerciales en caso de que el **equipo de desarrollo** sea **menor o igual a cinco personas**.

Para la realización del programa de seguimiento se ha utilizará la librería **OpenNI2** la cual incluye una licencia Apache 2.0 [73] por lo que en caso de distribuir junto con el programa de seguimiento el código de la librería se tendrán que tener en cuenta los siguientes aspectos:

- Se deberá incluir una copia de la licencia Apache 2.0
- Se deberá atribuir al autor de la librería original, en este caso la licencia está bajo el nombre de PrimeSense Ltd.
- Permite hacer uso del software de forma comercial y no comercial

El código de la librería OpenNI2 también incluye código de otra librería LibJPEG cuyos términos también se tendrán que tener en cuenta. Los aspectos de la licencia que afectaran al proyecto de forma más significativa serán:

- Se deberá incluir una copia de la licencia, y en caso de modificar la librería se deberá indicar de forma clara los cambios en la documentación
- En caso de solo distribuir un ejecutable se deberá añadir el la documentación "this software is based in part on the work of the Independent JPEG Group".
- Permite hacer uso del software de forma comercial y no comercial

OpenNI2 no accede directamente a las funciones de Kinect si no que hace uso del SDK de Microsoft para Kinect, por lo que **se tendrá que tener en cuenta** también **la licencia asociada al SDK oficial de Kinect,** v1.8. Los aspectos de la licencia que afectaran al proyecto de forma más significativa serán:

- El programa de instalación del SDK deberá ser el oficial proporcionado por Microsoft y se deberá testear el producto para que se ejecute en sistemas operativos Microsoft Windows.
- Hay un **uso restringido de sensores Kinect Xbox 360**, los sensores Kinect Xbox 360 solo pueden usados en consolas Xbox 360 sin embargo permite el uso de estos sensores para el desarrollo de aplicaciones que operen con Kinect para Windows. Por lo tanto los usuarios de la aplicación no podrían usar sensores Kinect 360 para ejecutar la aplicación no pudiéndose recomendar tampoco el uso de estos como alternativa a Kinect para Windows.
- **No está permitido crear un software de alto riego**, es decir que pueda provocar daños severos a una persona o dañar el medio ambiente (e.g navegación aérea, control de transportes de humanos). Aunque no debería haber ningún problema con este apartado, se ha tenido en cuenta y el controlador físico con el que interactúa el jugador es de gomaespuma evitando así posibles daños físicos o materiales, los marcadores de los extremos están creados con goma eva no con cartulina evitando el hipotético caso de cortes producidos por esta.
- **Permite el uso comercial y no comercial** siempre que se use la cámara Kinect para Windows.
- En caso de distribución **no está permitido** que la aplicación se pueda ejecutar en otros sistemas operativos diferentes a Microsoft Windows.

Otra librería a usar es **OpenCV** la cual tiene asociada una licencia BSD (Berkeley Software Distribution) la cual:

- Permite el desarrollo de aplicaciones comerciales y no comerciales
- Debe incluir un aviso de copyright el cual será incluido en el código del programa
- Hay una excepción en la licencia, en caso de usar las técnicas de visión artificial SURF y SIFT las cuales están patentadas y no permite su uso en aplicaciones comerciales. En el caso del programa de seguimiento se utilizará un filtro HSV por lo que no habrá problemas con este apartado.

Para la recolección de datos HSV máximos y mínimos dentro del programa de seguimiento se ha hecho uso de un método creado por Kyle Hounslow.

La licencia asociada al código de Kayle indica que se deberá incluir un aviso de copyright en el código, en concreto el mismo aviso que se deberá introducir al usar OpenCV. Aunque no esté especificado se atribuirá el método a Kayle Hounslow indicando en el código el nombre del autor junto con un enlace al código original [74].

Marco regulador de juego

Para el desarrollo del juego se ha utilizado el motor de videojuegos **Unity 5 Personal Edition** el cual permite el uso del motor de forma gratuita en aplicaciones no comerciales. Si la aplicación fuera a comercializarse Unity permitiría su **uso gratuito** mientras no se superara una cota de

ingresos brutos de 100.000 $ durante el año fiscal más reciente, en caso de superarlos se deberá adquirir una licencia para la versión Unity Proffesional Edition.

Para la conexión UDP de Unity se ha utilizará el código de Hasan Azizul [75] la cual está bajo una licencia WTFPL (Do What the Fuck You Want to Public License) la cual es altamente permisiva, permitiendo el uso del código tanto en aplicaciones no comerciales como comerciales además de no ser necesario realizar atribuciones. Sin embargo se cree oportuno indicar en el código el autor y el link del proyecto original.

El SkyBox (cielo) utilizado en el proyecto es un recurso creado por terceros que permite su **uso comercial y no comercial**.

Para la creación de sonidos en los primeros prototipos se ha hecho uso de la herramienta as3sfxr [76] la cual permite la creación de efectos sonoros de forma gratuita.

Para la creación de los sonidos del jugador se utilizará el programa **Audacity**, el cual permite la grabación y edición de audio de forma gratuita pudiendo ser usado en proyectos comerciales o no comerciales.

Para la implementación de sonidos del entorno junto con la textura de rotura de suelo se utilizará un pack de recursos del cual me hayo en posesión, la licencia de dicho pack permite el uso comercial y no comercial siempre y cuando el juego se distribuya de forma que no sea posible la extracción de los recursos del pack a partir del juego. Para solucionar dicha restricción se creará un ejecutable a la hora de distribuir el juego. Para más información sobre la licencia consultar [77].

Para el modelado 3D del coliseo y del mango del sable láser se hará uso del software gratuito **Blender** el cual usa una licencia GNU GPL (General Public License) versión 2 o posterior la cual permite su uso para aplicaciones comerciales y no comerciales. Aunque algunos módulos están bajo otro tipo de licencias (e.g Apache 2.0 [73] para Blender Cycles) estos no se utilizarán.

Entorno socioeconómico y presupuesto

Entorno socioeconómico

Aunque no se tiene intención de sacar el producto a la venta se ha considerado de interés hacer un breve análisis del entorno socioeconómico.

Videojuegos

Es un hecho que la industria del sector del videojuego está al alza, en España en concreto se incrementó la facturación un 8.7% comparado con 2014 alcanzando la cifra de los 1083 millones de euros, siendo las ventas físicas de 791 millones de euros y las ventas online 292 millones [78].

Visión artificial

Uno de los problemas existentes en la actualidad es la **abundancia de datos** en forma de imágenes o videos que necesitan ser procesados, una de las soluciones más viables es el uso de técnicas de visión artificial para procesarlas.

El ámbito en el que se puede aplicar el uso de visión artificial **es muy amplio**, desde seguridad, ocio, marketing etc… y por lo tanto cada vez hay más empresas dedicándose a investigar nuevas maneras de utilizar técnicas de visión artificial para mejorar sus productos. Un ejemplo

de esto es el amplio avance que se está consiguiendo en el entorno de los coches autónomos, mientras que Google se enfoca en el uso de radares LIDAR para formar modelos 3D, Tesla Motors cree que a pesar de la complejidad del problema el futuro de los coches autónomos se basará en el **uso de cámaras** con las que mediante técnicas de visión artificial se extraigan los datos del entorno necesarios, Mobileye uno de los mayores proveedores de hardware para coches autónomos parece apoyar también este enfoque [79].

George Hotz un famoso hacker estadounidense también comparte esta visión llegando a retar a Elon Musk (Tesla Motors) a que podría encontrar una solución mejor a la proveída por Movileye [80], la intención de Hotz es entrenar una red neuronal para que con un equipo inferior a 1000 dólares basado en cámaras de bajo coste y técnicas de visión artificial cualquier coche pueda conducirse de forma autónoma. Aunque aún esté en proceso los resultados son bastante sorprendentes [81].

Otro factor bastante relevante es la **accesibilidad a librerías de visión artificial gratuitas** como OpenCV en constante actualización, que permiten que varios tipos de usuarios puedan investigar sobre técnicas de visión artificial y aplicarlas a sus propios proyectos de una forma sencilla y gratuita.

Realidad virtual

El mundo de la realidad virtual está en auge estos últimos años, siendo cada vez más las empresas que apoyan la realidad virtual y sacan dispositivos dedicados a esta como por ejemplo Sony, Valve, Oculus incluso Microsoft la cual ha anunciado una nueva consola para 2017 con especificaciones para el uso de realidad virtual [82].

La salida al mercado en 2016 de la versión del consumidor de Oclulus Rift y HTC vive junto con la salida en octubre de las gafas de realidad virtual de Sony han propiciado que varios analistas hagan predicciones sobre las **ganancias que generará la realidad virtual**. Deloitte Global estima que solo en hardware las ventas en este año ascenderán a 700 millones de dólares [83], según Digi-Capital se estima que en 2020 la realidad virtual generará alrededor de 30 mil millones de dólares [84].

Presupuesto proyecto

Costes directos

Recursos humanos

Utilizando varias ofertas de empleo proporcionadas por InfoJobs se estima que el salario medio bruto de contratar a un ingeniero informático recién licenciado es de 1500 euros mensuales a jornada completa.

Si se considera el trabajo a jornada completa 20 días al mes se obtiene que el salario bruto será de 9.375 euros por hora.

Aunque el proyecto está definido entre el 1 de febrero y el 22 de junio del año 2016 se utilizará la normativa del trabajo de fin de grado para definir las horas con las que se hará el presupuesto. La normativa establece que para un trabajo de fin de grado el alumno debería invertir 300 horas [72], por lo tanto el coste de contratación será de **2812.5 euros.**

Hardware

Ordenador para desarrollo y pruebas 1200 €

Cámara Kinect V1 para Windows 0 €, prestadas para el desarrollo del proyecto por el departamento de informática de la universidad Carlos III, su coste en caso de tener en cuenta una fase de mantenimiento del producto sería de 169 € aproximadamente [85].

Gafas Oculus Rift DK1 0 €, prestadas para el desarrollo del proyecto por el departamento de informática de la universidad Carlos III. Las gafas actualmente no se encuentran a la venta de forma oficial por lo que se tendrían que adquirir de segunda mano, el coste estimado sería de 250€.

Mando Xbox 360 con receptor inalámbrico 50€

Software

El sistema operativo utilizado para el desarrollo y pruebas ha sido **Windows 10** con un coste de 0€, adquirido mediante DreamSpark [86] mediante el acceso como alumno de la Universidad Carlos III de Madrid.

Unity 5 Personal Edition 0€. En caso de comercializarlo y de obtener unos ingresos superiores a 100.000 $ se debería adquirir la edición Proffesional por 1.500$ o por 75$/mes.

Visual Studio Community 2015 0€. Se podría comercializar sin coste alguno debido a que en el desarrollo del producto no han participado más de 5 personas.

Pack **Skybox** (cielo) 0€

Pack Texturas y Sonidos 9,76€, para el desarrollo de sonidos ambientales y la textura de rotura del suelo se usará un paquete de recursos anteriormente adquirido.

Otros programas y librerías de uso gratuito para uso comercial y no comercial dadas las características del producto. **Blender**, **Audacity**, **OpenCV**, **OpenNI2**, **KinectSDK**, **As3sfxr**.

Otros

Churro de piscina 1.20 €

Goma eva lisa 0.6€ x2

Resumen costes directos

Nombre	Coste	Coste imputable
Ingeniero Informático	2812,5	2812,5
Ordenador	1200	96
Kinect V1 para Windows	0	0
Oculus Rift DK1	0	0
Mando Xbox 360 con receptor	50	4
Windows 10	0	0
Unity 5 Personal Edition	0	0
Visual Studio Community 2015	0	0
Pack Skybox	0	0

Pack Texturas y Sonidos	9,76	0,79
Blender	0	0
Audacity	0	0
OpenCV	0	0
OpenNI2	0	0
KinectSDK	0	0
As3sfxr	0	0
Churro de piscina	1,2	1,2
Goma eva lisa	0,8	0,8
Total		2915,29€

Otros costes

Costes indirectos

Los **costes indirectos** se corresponden con los gastos de electricidad, agua y utilización de material fungible. Se ha estimado que comprenderán el **10% del coste total directo**.

Riesgo

Ya que gran parte del entorno de desarrollo y tipo de proyecto es nuevo para el personal se ha considerado necesario incluir un coste añadido por el **riesgo** del proyecto del **12% sobre el coste total directo**.

Beneficios

El proyecto se **no se distribuirá de forma comercial** por lo que no existirá un porcentaje de beneficios. En caso de distribuirlo se añadiría un **10% sobre el coste total sin aplicar el riesgo** (costes directos+ indirectos).

Resumen total

Nombre	Coste
Costes directos	2915,29
Costes indirectos (10%)	291,52
Riesgo (12%)	349,9
TOTAL sin IVA	3556,71
IVA (21%)	747
TOTAL con IVA	4303,71€

El presupuesto total final es de **4303,71 €.**

5.Análisis y Diseño del problema

Requisitos

Descripción tabla requisitos

Identificador	
Nombre	
Prioridad	
Descripción	

Identificación: identificador del requisito, el formato será RX-Y-Z, siendo R la abreviación de requisito. X podrá tomar los valores F o NF (funcional, no funcional). El campo Y podrá tomar el valor PS en caso de que el requisito este comprendido dentro del sistema del programa de seguimiento o J en caso del juego. El campo Z será un número entero en el caso de los requisitos del programa de seguimiento y los requisitos no funcionales, en el caso de los requisitos funcionales de juego Z estará compuesto por una abreviación de la sección asignada al requisito y un número entero.

Organización y abreviaturas requisitos funcionales juego

Recepción y procesamiento de datos (RP)

Jugabilidad (J)

- Jugador (JU)
- Robot enemigo (R)
- Sable láser (SL)
- Entorno (E)

Gráficos (G)

Audio (A)

Nombre: nombre identificativo del requisito

Prioridad: en este campo se pueden dar dos valores Alta o Media en función de la importancia del requisito para el funcionamiento del programa.

Descripción: breve narración del requisito

Requisitos funcionales programa de seguimiento

Identificador	RF-PS-1
Nombre	Mostrar consola
Prioridad	Alta
Descripción	El programa de seguimiento deberá mostrar la ventana de la consola de comandos donde el usuario podrá introducir datos y se le mostrará la información pertinente sobre la ejecución o estado del programa de seguimiento.

Identificador	RF-PS-2
Nombre	Video RGB
Prioridad	Alta
Descripción	El programa de seguimiento deberá mostrar una ventana con el video a color en formato RGB resultado de la extracción de datos de la cámara Kinect.

Identificador	RF-PS-3
Nombre	Recolección datos HSV
Prioridad	Alta
Descripción	El programa de seguimiento contará con un sistema que permita obtener y almacenar los valores HSV mínimos y máximos de una región seleccionada por el ratón utilizando la ventana de video RGB.

Identificador	RF-PS-4
Nombre	Mostrar datos HSV
Prioridad	Media
Descripción	Al ser obtenidos los valores HSV mínimos y máximos se mostraran en la consola en forma de texto.

Identificador	RF-PS-5
Nombre	Cambio marcador
Prioridad	Alta
Descripción	El programa de seguimiento permitirá almacenar hasta dos conjuntos de valores HSV mínimos y máximos, permitiendo alternar la recolección de datos de uno a otro utilizando el ratón.

Identificador	RF-PS-6
Nombre	Filtros reducción de ruido
Prioridad	Alta
Descripción	El programa de seguimiento deberá contar con filtros de reducción de ruido para reducir posibles falsos positivos (reconocimiento de marcadores que no lo son como por ejemplo objetos de color similar)

Identificador	RF-PS-7
Nombre	Video imagen binaria fusionada
Prioridad	Media
Descripción	El programa de seguimiento deberá exponer el video de la imagen binaria, el cual contendrá el resultado de aplicar el filtro de color HSV de los dos marcadores junto con los filtros de reducción de ruido.

Identificador	RF-PS-8
Nombre	Puntero con coordenadas
Prioridad	Media

Descripción	En caso de encontrar un marcador se deberá mostrar en la pantalla de video a color un puntero que señale el centro del marcador junto al valor numérico de las coordenadas correspondientes a su ubicación dentro de la matriz de datos.

Identificador	RF-PS-9
Nombre	Dibujar bordes marcador
Prioridad	Media
Descripción	En caso de encontrar el marcador, se mostraran sus bordes en la ventana de video RGB, el color deberá ser diferente al del puntero y coordenadas.

Identificador	RF-PS-10
Nombre	Dibujar rectángulo selección
Prioridad	Alta
Descripción	A la hora de crear el rectángulo mediante el ratón para obtener los valores HSV mínimos se deberá mostrar en la pantalla de video RGB con un color diferente al del puntero, coordenadas y bordes del marcador.

Identificador	RF-PS-11
Nombre	Extracción coordenadas marcador
Prioridad	Alta
Descripción	El programa deberá ser capaz de extraer a partir de la localización del centro del marcador y el mapa de profundidad las coordenadas x, y, z en milímetros del marcador

Identificador	RF-PS-12
Nombre	Envío de coordenadas
Prioridad	Alta
Descripción	El programa de seguimiento deberá enviar poder enviar las coordenadas x, y, z de cada marcador al juego mediante la utilización de un puerto UDP.

Identificador	RF-PS-13
Nombre	Envío anterior de coordenadas
Prioridad	Alta
Descripción	En caso de que el programa de seguimiento no detecte un marcador deberá enviar los valores anteriores de este.

Identificador	RF-PS-14
Nombre	Filtro profundidad
Prioridad	Media
Descripción	El programa de seguimiento deberá contar con un filtro que evite enviar datos atípicos sobre la profundidad de los marcadores, se considerará dato atípico aquel en la que la profundidad del marcador difiera de 1 metro respecto a su anterior resultado, en cuyo caso se enviará el resultado del anterior valor.

Identificador	RF-PS-15
Nombre	Filtro distancia mínima
Prioridad	Media
Descripción	En caso de que el valor de la profundidad (z) para cualquiera de los 2 marcadores sea 0.8 metros o inferior no se enviará ningún mensaje.

Identificador	RF-PS-16
Nombre	Orden ventanas
Prioridad	Media
Descripción	Todas las ventanas deberán mostrarse de forma ordenada sin que una solape a otra.

Identificador	RF-PS-17
Nombre	Salir programa seguimiento
Prioridad	Alta
Descripción	El programa de seguimiento deberá permitir cerrar la aplicación utilizando el teclado, liberando en el proceso los recursos utilizados.

Identificador	RF-PS-18
Nombre	Interrumpir video RBG y binario
Prioridad	Media
Descripción	El programa de seguimiento deberá permitir interrumpir o reanudar mediante el teclado la visualización de datos de la ventana de video RGB y la ventana de video de la imagen binaria.

Identificador	RF-PS-19
Nombre	Reanudar video RBG y binario
Prioridad	Media
Descripción	El programa de seguimiento deberá permitir interrumpir o reanudar mediante el teclado la visualización de datos de la ventana de video RGB y la ventana de video de la imagen binaria.

Identificador	RF-PS-20
Nombre	Impresión valores teclado
Prioridad	Media
Descripción	Los valores insertados por teclado se mostrarán en la consola

Identificador	RF-PS-21
Nombre	Mostrar inicialización
Prioridad	Media
Descripción	El programa mostrará por texto en la consola el proceso de inicialización del programa

Identificador	RF-PS-22

Nombre	Mostrar instrucciones
Prioridad	Media
Descripción	El programa mostrará por texto en la consola las instrucciones necesarias para utilizar el programa.

Requisitos funcionales juego

Recepción y procesamiento de datos

Identificador	RF-J-RP-1
Nombre	Recepción datos
Prioridad	Alta
Descripción	El juego deberá poder recibir los datos enviados por el programa de seguimiento mediante el puerto UDP.

Identificador	RF-J-RP-2
Nombre	Liberar recursos
Prioridad	Alta
Descripción	Al cerrar o reiniciar la aplicación se deberán liberar los recursos utilizados.

Identificador	RF-J-RP-3
Nombre	Máximos y mínimos coordenadas
Prioridad	Alta
Descripción	El juego deberá tener unos valores máximos y mínimos para los valores obtenidos de las coordenadas, tanto para el movimiento del sable láser dentro del juego como de los valores recibidos del programa de seguimiento.

Identificador	RF-J-RP-4
Nombre	Aproximación coordenadas
Prioridad	Alta
Descripción	El juego deberá transformar aquellas coordenadas que sean mayores/menores a los máximos/mínimos definidos aproximándolas al valor máximo/mínimo más cercano.

Identificador	RF-J-RP-5
Nombre	Normalización coordenadas
Prioridad	Alta
Descripción	El juego deberá implementar un método que normalice las coordenadas enviadas por el programa de seguimiento al espacio de juego.

Identificador	RF-J-RP-6
Nombre	Movimiento sable láser
Prioridad	Alta
Descripción	El juego deberá procesar las coordenadas de los dos marcadores normalizadas y transformarlas en la rotación y traslación de la espada láser.

Jugabilidad

Jugador

Identificador	RF-J-J_JU-1
Nombre	Personaje
Prioridad	Alta
Descripción	El jugador contará con un personaje dentro del juego con el cual interactuará con el entorno.

Identificador	RF-J-J_JU-2
Nombre	Visión realidad virtual
Prioridad	Alta
Descripción	Se implementará realidad virtual en el juego mediante el casco de realidad virtual permitiendo al jugador observar el entorno virtual con el movimiento del casco.

Identificador	RF-J-J_JU-3
Nombre	Movimiento
Prioridad	Alta
Descripción	El jugador podrá mover al personaje por el entorno con el joystick izquierdo de un mando para Xbox 360, con las teclas w, a, s, d o con las flechas del teclado.

Identificador	RF-J-J_JU-4
Nombre	Rotación
Prioridad	Alta
Descripción	El jugador podrá rotar al personaje utilizando la rotación sobre el eje Y del casco de realidad virtual.

Identificador	RF-J-J_JU-5
Nombre	Caída
Prioridad	Alta
Descripción	El personaje se verá sometido por una fuerza de gravedad cayendo en caso de no contar con un apoyo.

Identificador	RF-J-J_JU-6
Nombre	Salto
Prioridad	Alta
Descripción	El jugador podrá hacer saltar al personaje utilizado el disparador izquierdo del mando para Xbox 360 o presionando la barra espaciadora del teclado.

Identificador	RF-J-J_JU-7
Nombre	Doble salto
Prioridad	Media
Descripción	El jugador podrá realizar otro salto en el aire habiendo transcurrido un tiempo mínimo de 0.5 segundos y menos de 2 segundos desde el primer salto.

Identificador	RF-J-J_JU-8
Nombre	Salto restricción
Prioridad	Alta
Descripción	Para realizar un salto el jugador deberá haber pisado anteriormente tierra, una vez gastado el salto o doble salto deberá tocar el suelo para volver a saltar otra vez.

Identificador	RF-J-J_JU-9
Nombre	Contador de vida
Prioridad	Alta
Descripción	El jugador tendrá un contador de vida con valor de 100 puntos.

Identificador	RF-J-J_JU-10
Nombre	Recentrar
Prioridad	Alta
Descripción	El jugador podrá recentrar al personaje en cualquier momento restableciendo a la rotación original de este utilizando el botón del joystick izquierdo o bien presionando la tecla "*c*" del teclado.

Identificador	RF-J-J_JU-11
Nombre	Aparecer robot
Prioridad	Alta
Descripción	El jugador podrá presionar la tecla "*h*" del teclado o el botón "*Select*" del mando Xbox 360 para hacer aparecer al robot.

Identificador	RF-J-J_JU-12
Nombre	Desaparecer robot
Prioridad	Alta
Descripción	El jugador podrá presionar la tecla "*h*" del teclado o el botón "*Select*" del mando Xbox 360 para hacer desaparecer al robot.

Robot enemigo

Identificador	RF-J-J_R-1
Nombre	Robot volador
Prioridad	Alta
Descripción	Se implementará un robot volador que dispare al personaje.

Identificador	RF-J-J_R-2
Nombre	Proyectil daño
Prioridad	Alta
Descripción	Se implementarán proyectiles que puedan dañar al personaje, cada proyectil que impacte en el cuerpo del jugador descontará 10 puntos a su vida.

Identificador	RF-J-J_R-3
Nombre	Proyectil tiempo vida

Prioridad	Alta
Descripción	El tiempo de vida máximo de un proyectil será de 5 segundos, una vez pasado el tiempo será destruido.

Identificador	RF-J-J_R-4
Nombre	Robot movimiento
Prioridad	Alta
Descripción	El robot se moverá por el entorno. Se definirá un intervalo de tiempo de 2 segundos, al terminar el robot decidirá si cambiar la dirección de movimiento de forma aleatoria.

Identificador	RF-J-J_R-5
Nombre	Rango movimiento
Prioridad	Alta
Descripción	Se definirá una serie de valores máximos y mínimos que defina el rango de movimiento para el robot.

Identificador	RF-J-J_R-6
Nombre	Corrección rango movimiento
Prioridad	Alta
Descripción	En caso de salirse del rango de movimiento el robot ignorará los cambios de dirección aleatorios y tratará de volver al rango rápidamente.

Identificador	RF-J-J_R-7
Nombre	Robot perseguidor jugador
Prioridad	Alta
Descripción	El rango de movimiento del robot será dependiente de la posición del jugador, viéndose sus valores modificados con el movimiento de este.

Identificador	RF-J-J_R-8
Nombre	Rango frontal robot
Prioridad	Alta
Descripción	El rango de movimiento del robot deberá ser tal que siempre quedará orientado a la parte frontal inicial del jugador.

Identificador	RF-J-J_R-9
Nombre	Robot disparo aleatorio
Prioridad	Alta
Descripción	Se definirá un intervalo de tiempo de 2 segundos en el que al terminar el robot decidirá si disparar al jugador de forma aleatoria.

Identificador	RF-J-J_R-10
Nombre	Robot rango disparo
Prioridad	Alta

Descripción	El robot no podrá disparar en caso de encontrase fuera del rango de movimiento.

Sable láser

Identificador	RF-J-J_SL-1
Nombre	Activación desactivación hojas sable láser
Prioridad	Alta
Descripción	El jugador podrá presionar la tecla "*l*" del teclado o el botón izquierdo "*lb*" del controlador Xbox 360 para activar la hoja superior del sable láser otra vez para activar la hoja inferior del sable láser y otra vez para desactivar ambas.

Identificador	RF-J-J_SL-2
Nombre	Deflactar proyectiles
Prioridad	Alta
Descripción	El jugador podrá deflactar los proyectiles con las hojas del sable láser, se implementará un campo de colisión que abarque la hoja de la espada.

Identificador	RF-J-J_SL-3
Nombre	Parar proyectiles
Prioridad	Alta
Descripción	El jugador podrá parar opcionalmente los proyectiles con el mango de la espada, en este caso no serán deflactados si no que desaparecerán.

Identificador	RF-J-J_SL-4
Nombre	Contar proyectiles deflactados
Prioridad	Media
Descripción	El jugador poseerá un contador de proyectiles deflactados con el sable láser, el contador se incrementará en caso de que el proyectil alcance el campo de colisión de una de hojas del sable láser.

Identificador	RF-J-J_SL-5
Nombre	Posicionar sable láser
Prioridad	Alta
Descripción	El sable láser siempre se mantendrá en la parte frontal original del personaje ignorando los cambios posteriores de rotación del jugador.

Entorno

Identificador	RF-J-J_E-1
Nombre	Posición inicial jugador
Prioridad	Alta
Descripción	La posición inicial del jugador y del robot estará aproximadamente en el centro del conjunto de placas.

Identificador	RF-J-J_E-2
Nombre	Placas selección
Prioridad	Alta
Descripción	Se implementará un intervalo de tiempo, transcurrido ese intervalo se seleccionaran un número aleatorio de placas para la caída de estas.

Identificador	RF-J-J_E-3
Nombre	Incremento dificultad placas
Prioridad	Alta
Descripción	La cantidad de placas seleccionadas para la caída deberá incrementarse a medida que transcurra el juego.

Identificador	RF-J-J_E-4
Nombre	Placas movimiento
Prioridad	Alta
Descripción	En caso de que una placa haya sido seleccionada se iniciará un contador el cual al llegar a 0 provocará la caída de la placa, transcurrido un tiempo las placas subirán de nuevo a su posición original, en el proceso las placas deberán atravesar el agua.

Identificador	RF-J-J_E-5
Nombre	Caída agua jugador
Prioridad	Alta
Descripción	Si el jugador cae al agua su contador de vida se actualizará a 0

Identificador	RF-J-J_E-6
Nombre	Transporte muerte
Prioridad	Alta
Descripción	Si el contador de vida del jugador llega a 0 el jugador morirá y será transportado a la superficie cristalina orientado al texto informativo.

Identificador	RF-J-J_E-7
Nombre	Estadísticas jugador muerte
Prioridad	Alta
Descripción	En caso de muerte el jugador podrá visualizar las estadísticas de juego (barra de vida, golpes recibidos y proyectiles deflactados) junto a estas se mostrará un texto que informará de la muerte del jugador, el tiempo de partida y el tiempo en el que se reanudará el juego

Identificador	RF-J-J_E-8
Nombre	Reinicio de juego por muerte
Prioridad	Alta
Descripción	Tras la muerte del personaje se reiniciará el juego tras 8 segundos

Identificador	RF-J-J_E-9

Nombre	Robot derrota
Prioridad	Alta
Descripción	El robot podrá ser alcanzado por uno de sus propios disparos al ser deflactado por el sable láser del jugador otorgando la victoria al jugador

Identificador	RF-J-J_E-10
Nombre	Transporte victoria
Prioridad	Alta
Descripción	En caso de victoria el jugador será transportado a la superficie cristalina orientado al texto informativo.

Identificador	RF-J-J_E-11
Nombre	Estadísticas jugador victoria
Prioridad	Alta
Descripción	En caso de muerte el jugador podrá visualizar las estadísticas de juego (barra de vida, golpes recibidos y proyectiles deflactados) junto a estas se mostrará un texto que informará de la victoria del jugador, el tiempo de partida y el tiempo en el que se reanudará el juego

Identificador	RF-J-J_E-12
Nombre	Reinicio de juego por victoria
Prioridad	Alta
Descripción	Tras la victoria del personaje se reiniciará el juego tras 10 segundos

Gráficos

Identificador	RF-J-G-1
Nombre	Composición suelo
Prioridad	Alta
Descripción	El suelo estará formado por dos tipos de placas de colores diferentes, estas se generarán al empezar el juego en forma de tablero de ajedrez.

Identificador	RF-J-G-2
Nombre	Placa aviso amarillo
Prioridad	Alta
Descripción	Cuando el contador para la caída de una placa se encuentre a la mitad se avisará al jugador cambiando el color de dicha placa a amarillo.

Identificador	RF-J-G-3
Nombre	Placa aviso rojo
Prioridad	Alta
Descripción	Cuando el contador para la caída de una placa esté a punto de finalizar y caiga se avisará al jugador cambiando el color de dicha placa a rojo.

Identificador	RF-J-G-4

Nombre	Placa restauración color
Prioridad	Alta
Descripción	Cuando la placa vuelva a su posición original se deberá restaurar el color original de la placa.

Identificador	RF-J-G-5
Nombre	Impacto proyectil suelo efecto
Prioridad	Media
Descripción	Si un proyectil impacta en el suelo instanciará una textura de rotura, junto con unas partículas de chispas y humo en la posición de colisión, al cabo de 3 segundos deberán desaparecer.

Identificador	RF-J-G-6
Nombre	Impacto proyectil hoja sable partículas
Prioridad	Media
Descripción	Si un proyectil impacta en una de las hojas del sable láser se instanciarán unas partículas de chispas y humo en la posición de colisión, al cabo de 3 segundos deberán desaparecer.

Identificador	RF-J-G-7
Nombre	Proyectil destrucción impacto
Prioridad	Alta
Descripción	El proyectil deberá destruirse tras impactar con cualquier objeto que no sea una de las hojas del sable láser.

Identificador	RF-J-G-8
Nombre	Fuegos artificiales partículas
Prioridad	Alta
Descripción	Al obtener la victoria se deberán instanciar partículas con forma de fuegos artificiales alrededor de la superficie cristalina.

Audio

Identificador	RF-J-A-1
Nombre	Aviso placa amarillo audio
Prioridad	Alta
Descripción	Cuando el contador para la caída de una placa se encuentre a la mitad se avisará al jugador mediante un sonido.

Identificador	RF-J-A-2
Nombre	Aviso placa rojo audio
Prioridad	Alta
Descripción	Cuando el contador para la caída de una placa finalice y caiga se avisará al jugador mediante un sonido.

Identificador	RF-J-A-3
Nombre	Aviso placa restauración audio
Prioridad	Alta
Descripción	Cuando la placa vuelva a su posición original se deberá avisar al jugador mediante un sonido.

Identificador	RF-J-A-4
Nombre	Salto audio
Prioridad	Alta
Descripción	Cuando el jugador presione el botón de salto se deberá escoger de entre cuatro sonidos de salto uno y reproducirlo.

Identificador	RF-J-J_A-5
Nombre	Aterrizaje audio
Prioridad	Media
Descripción	Cuando el personaje aterrice en el suelo se deberá escoger de entre tres sonidos de aterrizaje uno y reproducirlo.

Identificador	RF-J-A-6
Nombre	Daño jugador audio
Prioridad	Alta
Descripción	Cuando el personaje reciba un impacto de un proyectil se deberá escoger de entre cinco sonidos de daño uno y reproducirlo.

Identificador	RF-J-A-7
Nombre	Impacto proyectil en jugador audio
Prioridad	Alta
Descripción	Cuando el proyectil impacte sobre el personaje se deberá escoger de entre cinco sonidos de impacto uno y reproducirlo.

Identificador	RF-J-A-8
Nombre	Impacto proyectil en suelo audio
Prioridad	Alta
Descripción	Cuando el proyectil impacte sobre el suelo se deberá escoger de entre tres sonidos de impacto uno y reproducirlo.

Identificador	RF-J-A-9
Nombre	Impacto proyectil en hoja sable audio
Prioridad	Alta
Descripción	Cuando el proyectil impacte sobre una de las hojas del sable láser se deberá escoger de entre cuatro sonidos de impacto láser uno y reproducirlo.

Identificador	RF-J-J_A-10
Nombre	Impacto proyectil en mango audio

Prioridad	Media
Descripción	Cuando el proyectil impacte sobre el mango del sable láser se deberá escoger de entre tres sonidos de impacto uno y reproducirlo.

Identificador	RF-J-A-11
Nombre	Robot disparo audio
Prioridad	Alta
Descripción	Cuando el robot dispare se deberá escoger de entre cuatro sonidos de disparo uno y reproducirlo.

Identificador	RF-J-A-12
Nombre	Zumbido sable láser
Prioridad	Media
Descripción	Cada hoja del sable láser deberá emitir un sonido constante en bucle en caso de estar activada.

Identificador	RF-J-A-13
Nombre	Rotación sable láser audio
Prioridad	Media
Descripción	Se calculará una velocidad de rotación del sable láser desde sus coordenadas anteriores a las actuales, en caso de superar un límite definido se deberá escoger de entre diez sonidos de movimiento de sable láser uno y reproducirlo en el supuesto de que no haya uno reproduciéndose en el momento.

Identificador	RF-J-A-14
Nombre	Viento audio
Prioridad	Media
Descripción	Se reproducirá el sonido del viento en todo el mapa, el volumen deberá alcanzar su máximo en la superficie cristalina superior al coliseo, disminuirá en función de la lejanía del personaje.

Identificador	RF-J-A-15
Nombre	Agua audio
Prioridad	Media
Descripción	Se reproducirá el sonido del agua siendo su máximo en el centro del coliseo, disminuirá en función de la lejanía del personaje.

Identificador	RF-J-A-16
Nombre	Fuegos artificiales audio
Prioridad	Alta
Descripción	Al obtener la victoria se deberá reproducir el sonido de fuegos artificiales

Requisitos no funcionales

Requisitos no funcionales programa de seguimiento

Identificador	RNF-PS-1
Nombre	Compatibilidad Kinect
Prioridad	Alta
Descripción	El programa de seguimiento deberá ser compatible con Kinect V1 para Windows

Identificador	RNF-PS-2
Nombre	Sistema operativo Windows
Prioridad	Alta
Descripción	El programa de seguimiento deberá ser desarrollado para el sistema operativo Microsoft Windows

Identificador	RNF-PS-3
Nombre	Procesamiento imágenes simple
Prioridad	Alta
Descripción	El método de obtención de coordenadas del programa de seguimiento deberá ser simple no requiriendo mucho tiempo en el procesamiento de las imágenes.

Identificador	RNF-PS-4
Nombre	Tiempo comunicación
Prioridad	Alta
Descripción	El tiempo de comunicación entre programa de seguimiento y el del juego debe de ser rápido.

Identificador	RNF-PS-5
Nombre	Alta tasa mensajes por segundo
Prioridad	Alta
Descripción	El juego deberá recibir una tasa de actualizaciones de posición fluida, deberán llegar por parte del programa de seguimiento entre 20 o 30 mensajes por segundo.

Identificador	RNF-PS-6
Nombre	Modo normal Kinect
Prioridad	Alta
Descripción	Se utilizará el modo normal de la cámara Kinect, es decir esta obtendrá datos de profundidad 0.8 metros a 4 metros.

Identificador	RNF-PS-7
Nombre	Seguridad controlador físico
Prioridad	Media
Descripción	El controlador físico con el que interactúe el jugador deberá ser resistente a golpes, deberá ser adecuado para evitar posibles daños al jugador o al mobiliario adyacente a la zona de juego.

Identificador	RNF-PS-8
Nombre	Idioma de programa de seguimiento
Prioridad	Media
Descripción	El idioma de los textos informativos del programa de seguimiento deberá ser el inglés.

Identificador	RNF-PS-9
Nombre	Independencia programa de seguimiento
Prioridad	Alta
Descripción	El programa de seguimiento podrá ejecutarse independientemente del juego

Requisitos no funcionales juego

Identificador	RNF-J-1
Nombre	Tasa de refresco fluida
Prioridad	Alta
Descripción	El juego deberá ser fluido, se tomará la medida de 60 frames por segundo en la ejecución del juego como resultado óptimo.

Identificador	RNF-J-2
Nombre	Ejecutable Juego no VR
Prioridad	Media
Descripción	Se creará un ejecutable del juego que no requiera la instalación de Unity. El ejecutable deberá contener el juego el cual no requiera la presencia de un dispositivo de realidad virtual.

Identificador	RNF-J-3
Nombre	Ejecutable Juego VR
Prioridad	Alta
Descripción	Se creará un ejecutable del juego que no requiera la instalación de Unity, el ejecutable deberá contener el juego el cual requiera la presencia de un dispositivo de realidad virtual.

Identificador	RNF-J-4
Nombre	Compatibilidad sin mando
Prioridad	Media
Descripción	El juego podrá ser usado sin la necesidad de un controlador de Xbox 360, todas las funciones del mando deberán ser accesibles mediante el teclado.

Identificador	RNF-J-5
Nombre	Simultaneidad teclado mando
Prioridad	Media

Descripción	Se podrá usar simultáneamente el teclado y el controlador de Xbox 360.

Identificador	RNF-J-6
Nombre	Idioma de juego
Prioridad	Media
Descripción	El idioma de los textos informativos del juego deberá ser el inglés.

Identificador	RNF-J-7
Nombre	Independencia juego
Prioridad	Alta
Descripción	El juego se podrá ejecutarse independiente del programa de seguimiento.

Identificador	RNF-J-8
Nombre	Salida imagen de juego
Prioridad	Alta
Descripción	El juego deberá mostrarse tanto en las gafas de realidad virtual como en la pantalla del ordenador

Identificador	RF-J-9
Nombre	Superficie trasporte jugador
Prioridad	Alta
Descripción	Se creara una superficie independiente del suelo de placas ubicada en el aire en la parte superior del coliseo con visibilidad del texto informativo.

Identificador	RNF-J-10
Nombre	Material cristalino
Prioridad	Media
Descripción	El material de la superficie a la que se transporta al jugador tras su muerte o victoria deberá asemejarse al cristal, permitiendo que el jugador pueda observar el coliseo a través de este.

Identificador	RF-J-11
Nombre	Coliseo con agua
Prioridad	Alta
Descripción	El entorno de juego será un coliseo, este tendrá en la parte inferior de las gradas un agujero el cual estará relleno hasta cierto punto de agua.

Identificador	RF-J-12
Nombre	Suelo placas
Prioridad	Alta
Descripción	Flotando encima del agua estará el suelo el cual será creado al empezar el juego. Ubicado en la parte central del agujero del coliseo estará compuesto de una serie de placas con forma de cubo.

Identificador	RF-J-13
Nombre	Texto informativo
Prioridad	Alta
Descripción	En la parte superior del coliseo estará ubicado en forma que sea visible para el jugador un texto con el contador de proyectiles deflactados con el sable láser, un contador de golpes recibidos por proyectiles y una barra de vida.

Identificador	RF-J-14
Nombre	Cielo
Prioridad	Alta
Descripción	El entorno de juego deberá contener un cielo (Skybox).

Identificador	RF-J-15
Nombre	Iluminación sol
Prioridad	Alta
Descripción	El entorno de juego deberá ser iluminado por una fuente de luz que simule al sol, esta deberá producir sombras en los objetos y estar posicionada de tal manera que el personaje cuente con una cantidad de luz adecuada.

Identificador	RF-J-16
Nombre	Halo hoja sable
Prioridad	Alta
Descripción	La hoja del sable láser deberá contar con un halo de luz verde que simule la energía emitida por este.

Identificador	RF-J-17
Nombre	Iluminación hoja sable
Prioridad	Alta
Descripción	La hoja del sable láser deberá contar con fuentes de iluminación de color verde simulando la luz emitida por la hoja en los objetos adyacentes.

Identificador	RF-J-18
Nombre	Halo proyectil
Prioridad	Alta
Descripción	Los proyectiles deberán contar con un halo rojo simulando la energía emitida por un láser.

Diseño arquitectónico

En esta sección se explicará el diseño físico y lógico que tendrá tanto el videojuego como el programa de seguimiento.

Arquitectura física

En esta sección de detallaran los componentes físicos del sistema propuesto

-**Ordenador** de desarrollo y de pruebas: El ordenador deberá ser capaz de ejecutar el programa de seguimiento y el juego de forma simultánea. Aunque el casco de realidad virtual Oculus DK1 no requiere unos requisitos de sistema específicos sí que recomienda que el ordenador sea de gama alta, siendo capaz de alcanzar altas tasas de refresco para generar una experiencia fluida.

Las características del ordenador utilizado serán las siguientes

Características ordenador de desarrollo y pruebas

Sistema operativo	Windows 10 64 bits
Procesador	I7-3770K @ 3.5Ghz (8 CPUs)
Memoria RAM	16 GB DDR3
Tarjeta gráfica	NVIDIA GeForce GTX 770 (4GB DDR5)

El ordenador será el encargado de procesar los datos obtenidos por la cámara Kinect mediante el **programa de seguimiento** y de ejecutar el juego el cual será realizado mediante el motor de videojuegos **Unity**.

-**HMD** (head mounted display): se utilizará el casco Oculus Rift DK1, las características del casco están detalladas en la sección del estado del arte.

-Cámara **Kinect** V1 para Windows: se utilizará para obtener las coordenadas de los 2 marcadores, las características de la cámara están detalladas en la sección del estado del arte.

-**Monitor**: en él se mostrará el juego y las ventanas de video del programa de seguimiento, la resolución es de 1680x1050

-**Altavoces** u otro dispositivo de audio: se utilizará para reproducir el audio del juego.

-**Controlador de gomaespuma**: se utilizará un cilindro de gomaespuma de color rojo con 2 trozos de goma eva de color verde y azul ubicados en los extremos de este, estos serán los marcadores a localizar.

-**Ratón**: se utilizará como alternativa en caso de querer captar otros colores diferentes al verde o al azul dentro del programa de seguimiento.

-**Teclado**: se utilizará como dispositivo de input tanto en el programa se seguimiento como en el juego.

-**Mando** de **Xbox 360** inalámbrico: se utilizará como dispositivo de input para el juego.

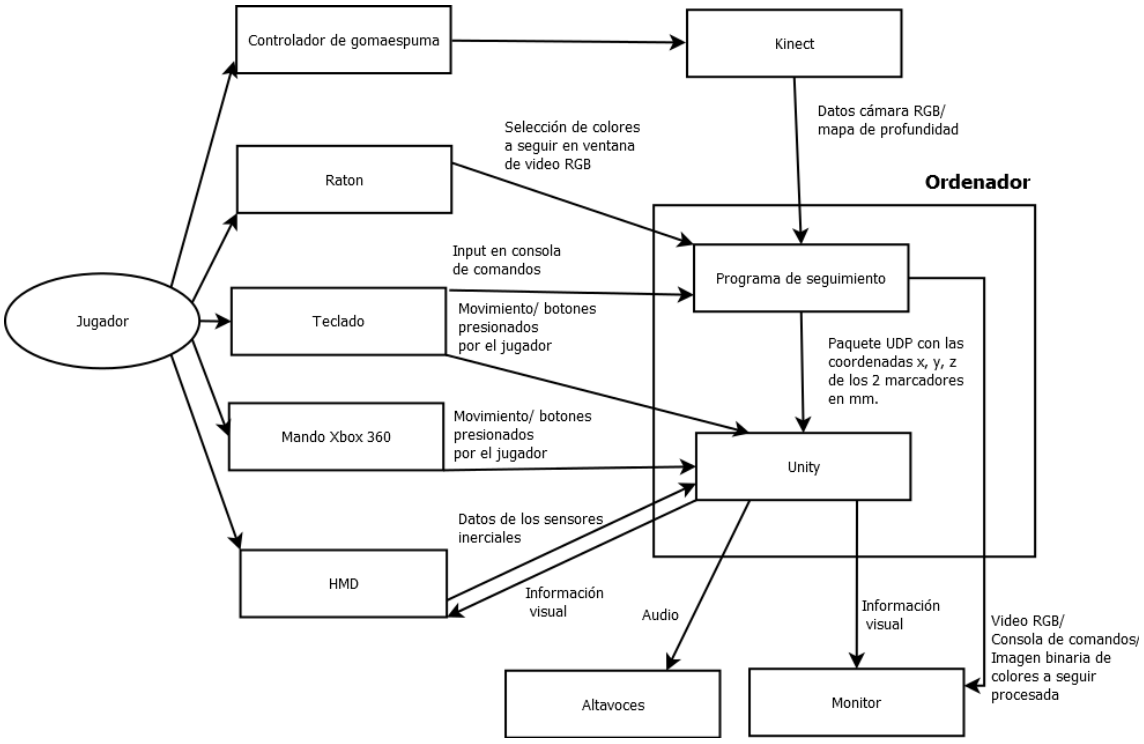


Diagrama de arquitectura física del proyecto

Arquitectura lógica

Casos de uso

Descripción de campos casos de uso

Identificador	
Nombre	
Actores	
Prioridad	
Precondiciones	
Descripción	

Identificación: identificador del caso de uso, el formato será CU-X-Y, siendo CU la abreviación de caso de uso. El campo X podrá tomar el valor PS en caso de que el caso de uso este comprendido dentro del sistema del programa de seguimiento o J en caso del juego. El campo Y será un número entero.

Nombre: nombre identificativo del caso de uso

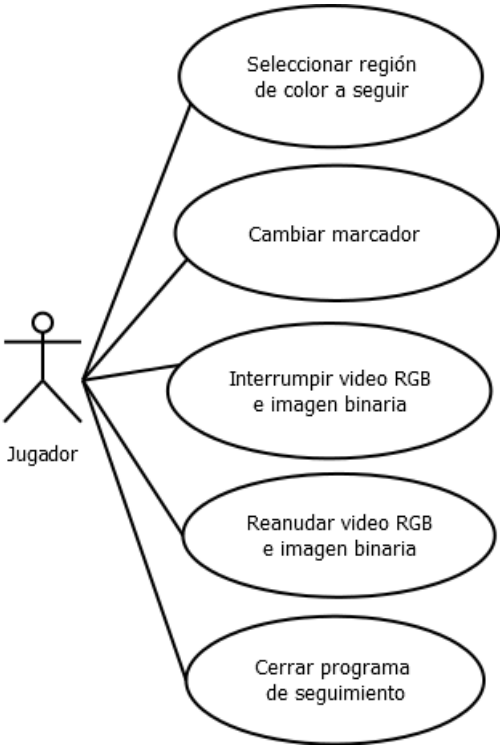
Actores: actores asociados al casos de uso.

Prioridad: en este campo se pueden dar dos valores Alta o Media en función de la importancia de la acción para la resolución del programa.

Precondiciones: se describirán las condiciones que se deben dar antes de poder ejecutar el caso de uso.

Descripción: breve narración del caso de uso.

Diagrama de casos de uso programa de seguimiento



Definición de actores programa de seguimiento

Solo se contará con un actor el cual será el **jugador**, el cual representará a la persona que ejecute el programa e interaccione con las diferentes opciones que este ofrece.

Casos de uso programa de seguimiento

Identificador	CU-PS-1
Nombre	Seleccionar región de color a seguir
Actores	Jugador
Prioridad	Alta
Precondiciones	-El programa debe de haber inicializado la cámara Kinect -El programa debe de haber abierto la ventana del video RGB
Descripción	El jugador podrá utilizar el click izquierdo del ratón en la ventana de video RGB y manteniéndolo pulsado generar un rectángulo en esta, tras soltar el click izquierdo se mostrará en la pantalla de la consola los valores HSV mínimos y máximos encontrados en el área seleccionada.

Identificador	CU-PS-2
Nombre	Cambiar marcador
Actores	Jugador
Prioridad	Alta
Precondiciones	-El programa debe de haber inicializado la cámara Kinect

	-El programa debe de haber abierto la ventana del video RGB
Descripción	El jugador podrá presionar el click derecho del ratón para cambiar el marcador del cual se vayan a recoger los datos HSV.

Identificador	CU-PS-3
Nombre	Interrumpir video RGB e imagen binaria
Actores	Jugador
Prioridad	Media
Precondiciones	-El programa debe de haber inicializado la cámara Kinect -El programa debe de haber abierto la ventana del video RGB -El programa debe de haber abierto la ventana de imagen binaria fusionada - El programa debe de haber procesado la imagen binaria, aplicando los filtros correspondientes y fusionando la imagen binaria de ambos marcadores. - El estado de mostrar los videos deberá estar definido a verdadero - El jugador deberá de haber presionado la tecla 'c'
Descripción	El programa deberá evitar mostrar nueva información en las ventanas de video RGB e imagen binaria fusionada, el usuario podrá acceder a dicha función presionando la letra correspondiente del teclado, el proceso de seguimiento de los marcadores no deberá verse afectado.

Identificador	CU-PS-4
Nombre	Reanudar video RGB e imagen binaria
Actores	Jugador
Prioridad	Media
Precondiciones	-El programa debe de haber inicializado la cámara Kinect -El programa debe de haber abierto la ventana del video RGB -El programa debe de haber abierto la ventana de imagen binaria fusionada - El estado de mostrar los videos deberá estar definido a falso - El jugador deberá de haber presionado la tecla 'c'
Descripción	El programa deberá mostrar la información en las ventanas de video RGB e imagen binaria fusionada, el usuario podrá acceder a dicha función presionando la letra correspondiente del teclado, el proceso de seguimiento de los marcadores no deberá verse afectado.

Identificador	CU-PS-5
Nombre	Cerrar programa de seguimiento
Actores	Jugador
Prioridad	Alta
Precondiciones	- El programa debe de haber inicializado la cámara Kinect - El jugador deberá de haber presionado la tecla 'e'
Descripción	El jugador cerrará el programa de seguimiento liberando los recursos utilizados en el proceso, el usuario podrá acceder a dicha función presionando la letra correspondiente del teclado

Diagrama de casos de uso juego

Definición de actores programa de seguimiento

Solo se contará con un actor el cual será el **jugador**, el cual representará a la persona que ejecute el juego e interaccione con las diferentes opciones que este ofrece.

Casos de uso juego

Identificador	CU-J-1
Nombre	Bloquear proyectil
Actores	Jugador
Prioridad	Alta
Precondiciones	-El juego debe de haberse inicializado -El robot debe de haber disparado algún proyectil

	-La hoja del sable láser o el mango debe colisionar con proyectil
Descripción	Se producirá el bloqueo al colisionar el proyectil con una de las dos hojas del sable láser o el mango. En caso de colisión con una hoja el proyectil será deflactado, en caso de que colisione en el mango el proyectil desaparecerá.

Identificador	CU-J-2
Nombre	Ganar
Actores	Jugador
Prioridad	Alta
Precondiciones	-El juego debe de haberse inicializado -El robot debe de haber disparado algún proyectil -El jugador debe de haber deflactado un proyectil con una de las hojas del sable -Un proyectil debe de colisionar con el robot
Descripción	El jugador podrá ganar la partida lo cual le transportara a una superficie cristalina en el cielo donde tendrá visión de los resultados de la partida y se le informará de su victoria, el juego deberá reiniciarse tras 10 segundos

Identificador	CU-J-3
Nombre	Recibir disparo
Actores	Jugador
Prioridad	Alta
Precondiciones	-El juego debe de haberse inicializado -El robot debe de haber disparado algún proyectil -El proyectil debe impactar en el jugador
Descripción	Si un proyectil impacta en el jugador se le descontaran 10 puntos de vida y se informará al jugador del impacto mediante el audio, la actualización de la barra de vida y el contador de golpes recibidos.

Identificador	CU-J-4
Nombre	Caer al agua
Actores	Jugador
Prioridad	Alta
Precondiciones	-El juego debe de haberse inicializado -El jugador debe de entrar en contacto con el agua
Descripción	El jugador podrá caer en el agua provocando que el contador de vida baje a 0

Identificador	CU-J-5
Nombre	Morir
Actores	Jugador
Prioridad	Alta
Precondiciones	-El juego debe de haberse inicializado -El jugador debe tener el contador de vida a 0
Descripción	El personaje podrá morir o bien por caída al agua lo cual implicará su muerte directa o bien al recibir un impacto de un proyectil que baje su vida a 0 en tal caso se deberá transportar al jugador a una superficie cristalina en el cielo donde tendrá visión de los resultados de la partida y se le informará de su derrota, el juego deberá reiniciarse tras 8 segundos

Identificador	CU-J-6
Nombre	Activar una hoja del sable láser
Actores	Jugador
Prioridad	Alta
Precondiciones	-El juego debe de haberse inicializado -Las dos hojas del sable láser deberán estar en estado desactivado -El jugador debe de haber presionado la tecla del mando Xbox360 “lb” o la tecla “l” del teclado
Descripción	El jugador activará la hoja superior del sable láser activando todas sus funciones

Identificador	CU-J-7
Nombre	Activar las dos hojas del sable láser
Actores	Jugador
Prioridad	Media
Precondiciones	-El juego debe de haberse inicializado -La hoja superior del sable láser deberá estar en estado activado -El jugador debe de haber presionado la tecla del mando Xbox360 “lb” o la tecla “l” del teclado
Descripción	El jugador activará la hoja inferior del sable láser activando todas sus funciones

Identificador	CU-J-8
Nombre	Desactivar hojas del sable láser
Actores	Jugador
Prioridad	Media
Precondiciones	-El juego debe de haberse inicializado -Las dos hojas del sable láser deberán estar en estado activado -El jugador debe de haber presionado la tecla del mando Xbox360 “lb” o la tecla “l” del teclado
Descripción	El jugador desactivara las dos hojas del sable láser desactivando todas sus funciones

Identificador	CU-J-9
Nombre	Hacer aparecer robot
Actores	Jugador
Prioridad	Alta
Precondiciones	-El juego debe de haberse inicializado -El robot deberá estar en estado desactivado -El jugador debe de haber presionado la tecla del mando Xbox360 “select” o la tecla “h” del teclado
Descripción	El jugador podrá activar el robot activando todas sus funciones

Identificador	CU-J-10
Nombre	Hacer desaparecer robot
Actores	Jugador
Prioridad	Media

Precondiciones	-El juego debe de haberse inicializado -El robot deberá estar en estado activado -El jugador debe de haber presionado la tecla del mando Xbox360 “select” o la tecla “h” del teclado
Descripción	El jugador podrá desactivar el robot desactivando todas sus funciones

Identificador	CU-J-11
Nombre	Saltar
Actores	Jugador
Prioridad	Alta
Precondiciones	-El juego debe de haberse inicializado -El jugador deberá haber tocado tierra anteriormente en caso de ser el primer salto -El jugador deberá de haber realizado un primer salto anteriormente para ejecutar el doble salto y haber trascurrido medio segundo desde este
Descripción	El jugador podrá ejecutar un salto provocando el movimiento del personaje hacia arriba ejecutando el audio de salto, el primer salto habilitará la opción de realizar un doble salto propulsando con la misma fuerza al personaje y ejecutando el audio de salto

Identificador	CU-J-12
Nombre	Mover personaje
Actores	Jugador
Prioridad	Alta
Precondiciones	-El juego debe de haberse inicializado -El jugador deberá haber movido el joystick izquierdo del mando de Xbox 360 o haber presionado las teclas “w” “a” “s” “d” o flechas del teclado
Descripción	El jugador será movido acorde con la información de los dispositivos de input, la posición del jugador deberá sumarse a la posición del sable láser para que este acompañe al jugador.

Identificador	CU-J-13
Nombre	Recentrar personaje
Actores	Jugador
Prioridad	Alta
Precondiciones	-El juego debe de haberse inicializado -El jugador deberá de haber hecho click en el joystick izquierdo del mado Xbox 360 o haber presionado la tecla “c” en el teclado
Descripción	La rotación del personaje volverá a su orientación original

Identificador	CU-J-14
Nombre	Rotar personaje
Actores	Jugador
Prioridad	Alta
Precondiciones	-El juego debe de haberse inicializado -El casco de realidad virtual debe de estar conectado -El jugador deberá de rotar la cabeza con el casco de realidad virtual

Descripción	El jugador rotará sobre el eje Y al personaje en función de hacia donde este orientado el caso, el sable láser no deberá verse afectado por la rotación.

6.Planteamiento de la solución

Este apartado está dividido en dos partes:

En el **desarrollo de la solución** se muestra el proceso que se pasó para dar con la solución final. Se irá explicando las diversas mecánicas implementadas, posibles alternativas y algunos de los problemas más relevantes encontrados.

En la **explicación de la solución** se explican más en detalle las principales funcionalidades del proyecto.

Desarrollo de la solución

El primer paso para buscar una herramienta con la que solucionar el proyecto fue intentar utilizar las funciones del SDK (software development kit) 1.8 de Kinect (última versión para la versión Kinect V1). Para ello se siguieron las instrucciones de [87]y [88] los cuales usaban WPF (Windows presentation foundation) en el IDE Visual Studio 2015 para crear una interfaz con la que se pudiera interaccionar con la mano. El objetivo de este tutorial permitiría **valorar si el SDK de Kinect era apto para desarrollar una solución** e introducirme en el IDE Visual Studio 2015.

Tras realizar el programa descubrí que aunque la cámara se mostraba como conectada no se visualizaba el contenido de la cámara tal y como indicaban los tutoriales. Traté pues de ejecutar el programa en Kinect for Windows Developver Toolkit v1.8 [89] (contiene varios programas ejecutables para Kinect, entre ellos el mostrado en [87] y [88]) para comprobar si el problema era de la cámara. Tras varias pruebas comprobé que la cámara funcionaba para todos los ejemplos excepto para el implementado y otro ejemplo para sustituir el fondo de detrás del usuario. Ya que no encontré la causa del error y no conseguí encontrar ningún método que me permitiera reconocer marcadores decidí **descartar el uso de SDK de Kinect** para elaborar la solución.

Se procedió entonces a instalar la librería **OpenNI2**, esta librería permitía acceder a funcionalidades de la cámara Kinect sin embargo la versión instalada **no accede directamente a los datos de la cámara** si no que **utiliza el SDK de Kinect** para acceder a ellos por lo que para su utilización es **necesario instalar el SDK de Kinect** oficial (última versión 1.8). El uso de OpenNI2 permitía además utilizar otras librerías de forma complementaria en caso de ser necesario, como por ejemplo la librería de visión artificial OpenCV

Para realizar la instalación de la librería en Visual Studio 2015 y para aprender las funcionalidades que ofrecía esta, se recurrió a la lectura de [90]. Se probaron y modificaron los ejemplos incluidos lo cual permitió aumentar el conocimiento de que funciones que podían utilizarse mediante Kinect.

Entre las funciones encontradas en [90] estaba la inicialización de la cámara Kinect y el acceso a sus datos, también enseñaba a como **alinear los resultados del mapa de profundidad con los de la cámara RGB** (las cámaras están en posiciones diferentes y por lo tanto si no se alinean los datos, los pixeles de una imagen no se corresponden con los de la otra). Otro ejemplo mostraba la posibilidad de grabación de un fichero de **video .ONI** el cual permitía almacenar los datos de la cámara RGB y el mapa de profundidad en un mismo video.

En [90] también se enseñaba a instalar y configurar la librería OPENGL (open graphics library) la cual era usada para mostrar datos por pantalla como por ejemplo el resultado de la cámara RGB o el mapa de profundidad. Los últimos capítulos ofrecían ejemplos también del funcionamiento de la librería middleware NITE la cual da acceso a varias funciones de seguimiento de esqueleto y manos del usuario. Aunque se leyeron algunos ejemplos no se hizo ninguna prueba ya que **el seguimiento de partes del cuerpo no era una de las prioridades del proyecto**, sin embargo ayudó a que se pensaran varias ideas de posibles ampliaciones del proyecto.

Una vez se tuvo la suficiente información se llegó a la conclusión de que **no era posible utilizar únicamente la librería OpenNI** para hacer el seguimiento de los marcadores por lo que se empezó a buscar información sobre otras bibliotecas.

Se valoraron librerías de visión artificial como AFORGE (C#) y el .NET wrapper EMGUCV (adaptador de OpenCV que permite usar lenguajes como C#). La ventaja usar una librería utilizando C# es que utilizaría el mismo lenguaje que Unity y aunque se desarrollara las dos partes por separado sería mucho más fácil utilizar un mismo lenguaje para todo el desarrollo. Sin embargo al final se escogió el uso de la librería de visión artificial **OpenCV** (open source computer vision).

¿Por qué usar OpenCV?

- Es una de las librerías de visión artificial más famosa y amplia
- **Es veloz y eficiente**. OpenCV es una librería escrita en C/C++ lo cual permite una **mayor optimización del código**. En el caso de nuestro proyecto es vital conseguir la mayor velocidad de tratamiento de imágenes posible ya que necesitamos hacer el seguimiento de los marcadores de la forma más fluida posible.
- **Su uso es gratuito** (licencia BSD), tanto para aplicaciones académicas como comerciales.
- Está **disponible para Windows**, Linux y MacOS, también puede ser utilizada en Android e iOS
- **Tiene una gran cantidad de documentación**, tanto en la página oficial como en varios libros.

Llegados a ese punto ya se tenía bastante claro que se iba a usar para cada parte del proyecto.

- Programa de seguimiento: uso de librerías **OpenNI** y **OpenCV** (C++) en el IDE **Visual Studio 2015**
- Juego: **Unity 5** (C#)

El siguiente problema a resolver era como se iba a realizar la **comunicación entre ellos**. Buscando información se encontró un video [75], en el que se mostraba una aplicación que reconocía mediante OpenCV aplicando la técnica CamShift la posición en dos dimensiones de una taza. Tras obtener la posición del objeto **enviaba las coordenadas a Unity** usando un **puerto UDP** de modo el juego moviera un cubo en consonancia.

Una vez se consiguió averiguar la manera de cómo comunicar el programa de seguimiento con el juego quedaba pensar en un **método para el seguimiento de los marcadores**. La idea principal era usar **dos marcadores de distinto color** de los que extraer sus coordenadas x, y, z. A partir de las coordenadas en el espacio 3D de dos puntos se podía extraer la **posición y rotación de un único objeto**, en este caso el de una espada láser.

La idea de hacer un seguimiento de colores además a primera vista parecía bastante más simple computacionalmente que cualquier otro sistema de seguimiento, como CamShift o la utilización de una nube de puntos (e.g uso de la librería PCL).

Otra opción similar que se consideró fue la de usar tres marcadores de un mismo color de manera que se pudiera saber tanto la orientación del objeto como la posición (con 2 del mismo color esto sería imposible), la ventaja que ofrecía era que al usar el mismo color sería más fácil de localizar los marcadores (si se usan distintos colores dependiendo de la iluminación puede que se detecte un color mejor o peor que el otro), se descartó para minimizar el número de marcadores.

Buscando información se encontró con el video de Kyle Hounslow [91], el cual explicaba cómo aplicar un filtro de un color, hacer el seguimiento y obtener las coordenadas 2D de dicho color.

Tratando de instalar la librería OpenCV para Visual Studio 2015 se encontró con un problema que a pesar de ser de fácil solución **retraso bastante el proceso**. A la hora de referenciar la librería esta se había insertado en una carpeta cuyo nombre tenía un espacio entre medias ("TFG librerías") lo cual no es una forma valida de referenciar una librería en Visual Studio (a no ser que la ruta este encerrada en comillas (e.g ""C:/TFG librerías/opencv"")). Se tardó bastante en comprender la naturaleza del error (se pensaba que era un problema de versiones o un problema de compatibilidad con OpenNI) lo cual retraso varios días el proyecto. Para **prevenir ese tipo de errores** y para **facilitar la instalación de la librería** se ha añadido un **anexo** al final de la memoria que indica el proceso de instalación paso por paso incluyendo la explicación de los errores más comunes para evitarlos.

Una vez instalado OpenNI y OpenCV se procedió a la definir los prototipos a realizar.

- **Protipo_1**: a partir de un video ser capaz de realizar el seguimiento de un color en el espacio 2D (extracción de coordenadas x, y) y de enviar las coordenadas a Unity.

- **Prototipo_2**: recibir en Unity los datos enviados por el programa de seguimiento, normalización de estos y aplicarlos para el movimiento de un cubo.

- **Prototipo_3**: a partir de un video .ONI (contiene información de la cámara RGB y del mapa de profundidad) ser capaz de realizar el seguimiento de un color en el espacio 3D (extracción de coordenadas x, y, z) y de enviar las coordenadas a Unity.

- **Prototipo_4**: utilizando la cámara Kinect realizar el seguimiento de un color en el espacio 2D y de enviar las coordenadas a Unity.

- **Prototipo_5**: utilizando la cámara Kinect realizar el seguimiento de un color en el espacio 3D y de enviar las coordenadas a Unity.

- **Prototipo_6**: utilizando la cámara Kinect realizar el seguimiento de dos colores en el espacio 3D y de enviar las coordenadas a Unity.

Para el desarrollo del **prototipo_1** se utilizó un video que enseñaba como hacer malabares [92] para intentar hacer un seguimiento de las pelotas. El objetivo de este prototipo era introducirse en OpenCV y sus funciones de reconocimiento de color. Además la utilización de un video permitía **realizar las pruebas de una forma más sencilla y rápida**.

Para la realización de la parte de reconocimiento de color se utilizó el proyecto de Kyle Hounslow [91] y la documentación oficial de OpenCV de la cual también se extrajo la información necesaria para trabajar con un video.

Para realizar él envió de información a Unity se planteó una pregunta, ¿se usaría el **protocolo TCP** o **UDP**?

El protocolo TCP garantiza que los datos lleguen al destino de una forma correcta mediante un control de recepción pero requiere de una conexión previa con el destinatario.

El protocolo UDP no garantiza la integridad de los datos pero no requiere de una conexión previa.

Analizando sus ventajas e inconvenientes se eligió el protocolo **UDP** ya que no importaba tanto que algunos datos llegaran de forma incorrecta si no que **primaba la velocidad de comunicación** entre ambas aplicaciones (debido a que un mal tiempo de respuesta en la espada puede provocar una mala sensación de control al jugador). Además salvo movimientos muy bruscos los demás datos tendrían gran cantidad de redundancia sobre la posición del controlador por lo que **no es problema que perdamos parte de la información**. La implementación de la comunicación por el puerto UDP se explica con más detalle, en la sección “Explicación de la solución”.

Resumen de funcionamiento **prototipo_1**: se abre el video, se inicia un evento por el cual se puede seleccionar en cualquier momento el color a seguir, en un bucle se extrae en cada iteración el fotograma correspondiente, se aplica un filtro al fotograma para que solo muestre como color blanco los pixeles que contuvieran el color seleccionado y negro para el resto, se aplican filtros para reducir el ruido y se selecciona el mayor conjunto de pixeles blancos colindantes. Posteriormente se calcula el centroide del conjunto de pixeles blancos siendo este las coordenadas x, y de nuestro objeto, se manda la información por el puerto UDP y se muestra al usuario un feedback (e.g puntero al objeto encontrado).

Para el desarrollo del **prototipo_2** se utilizó mayoritariamente el código del video [75], el cual permitía recibir datos mediante un puerto UDP. También se hizo una **normalización** de las coordenadas recibidas para limitar el movimiento del cubo a la parte visible de la pantalla del juego.

En la siguiente imagen se muestra el **prototipo_1** con el **prototipo_2** en funcionamiento.

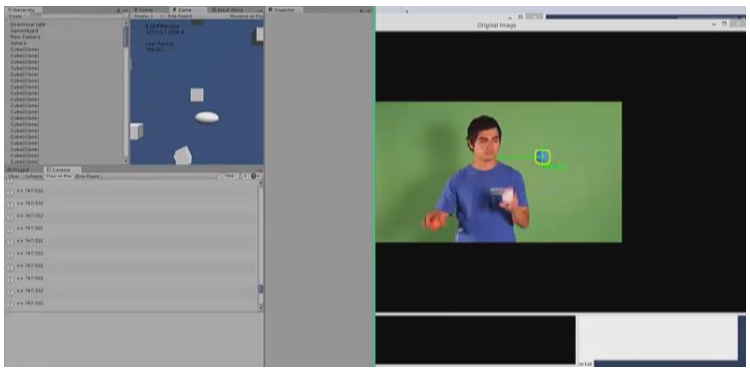


Ilustración 31 Funcionamiento prototipo_1 y prototipo_2

La creación del **prototipo_3** tenía como objetivo hacer uso del mapa de profundidad para calcular la distancia a la que se encontraba el marcador de color, para hacer esto era necesario

usar un video .ONI simulando la cámara Kinect. Para la grabación del video .ONI se utilizó NiViewer una aplicación instalada con OpenNI que permitía grabar videos .ONI.

Ilustración 32 NiViewer en funcionamiento

El **problema** es que a pesar de **abrir el video** y acceder a los **datos de ambas cámaras** NIR y RGB, **no era posible** ejecutar la **función de alineado** de OpenNI en un video (ya que las cámaras están en posiciones distintas los pixeles del mapa de profundidad no se corresponden con datos de la cámara RGB). Debido a este problema se **descartó el uso de videos** para probar el funcionamiento del programa

Para el desarrollo del **prototipo_4** se intentó usar el programa CMake [93] para compilar la librería OpenCV con compatibilidad OpenNI y habilitar nuevas funciones [94] que permitieran acceder a la cámara Kinect con funciones OpenCV. Sin embargo las primeras pruebas que se realizaron usando este método generaron bastantes problemas por lo que se decidió usar las funciones de ambas bibliotecas de forma **independiente**.

Este prototipo utilizó casi todas las funcionalidades ya existentes del **prototipo_1**, exceptuando que esta vez los datos no provenían de un video si no de la cámara Kinect a la cual se accedía mediante OpenNI, para la realización del acceso y configuración de la cámara Kinect se volvió a consultar el libro [90].

Para las **pruebas del prototipo_4** se probó a realizar el seguimiento con **varias pelotas de distintos tamaños y colores**. Se percibió que la **iluminación influía** bastante en la **calidad del reconocimiento del color** ya que dependiendo de cómo incidiera la luz sobre la pelota la cámara captaría un color u otro. Otro problema fue la aparición de un error que provocaba que el programa de seguimiento se cerrara en el minuto 1:30, esto solo sucedía cuando se había abierto anteriormente el editor de Unity. Se intentó arreglar reinstalando el editor pero el fallo se mantuvo. Tras varias pruebas se creó un ejecutable con el juego para hacer pruebas más largas a un minuto y medio ya que de esta manera no era necesario abrir el editor**, no se consiguió averiguar el origen del error** de modo que se anota para su posterior corrección.

En el **prototipo_5** se descubrió que el **modo normal** de la cámara Kinect solo aceptaba calcular profundidad a partir de **0.8m** (Kinect para Windows V1 permite mediante el **modo cercano** hasta 0.5m sin embargo no se consideró necesario implementarlo). Esto provocó que las pruebas no se pudieran realizar con objetos pequeños como los que se llevaban utilizando hasta el momento, de modo que para las pruebas se usaron varios marcadores más grandes hechos con cartulina ubicados a distintas distancias.

Resumen de funcionamiento **prototipo_5**: se abre la cámara, se configura la resolución a la que grabará, a continuación **se alinea** el **mapa de profundidad con** la imagen de la **cámara RGB**. Se consiguen las coordenadas x e y a través de las funciones ya desarrolladas en el **prototipo_4**, con dichas coordenadas se accede al pixel correspondiente del mapa de

profundidad y se extrae la distancia en milímetros de este, por último se envían las coordenadas x, y, z por el puerto UDP.

Para realizar las pruebas se necesitó actualizar el **prototipo_2**, modificando la función de extracción de datos del puerto y la función de normalización para poder **normalizar** el valor de la coordenada **z**.

La implementación del **seguimiento de un segundo color** en el **prototipo_6** no fue tan difícil como se esperaba, aunque tras las pruebas realizadas en el anterior prototipo se decidió implementar un **filtro** que evitara mandar mensajes posiblemente erróneos. Si de un mensaje a otro hay 1 metro de diferencia o si los marcadores se encontraban a menos de 0.8 metros no se enviaría el mensaje.

Para las **pruebas** del **prototipo_6** se creó un controlador para simular el sable láser. Se compró un churro de piscina (el material permitía evitar daños al jugador, personas adyacentes o al mobiliario) y goma eva lisa de distintos colores a modo de marcadores (la cual también amortiguaba posibles golpes).

Se eligió el color **rojo** como color principal del controlador de gomaespuma para contrastar con el **verde** y **azul** de los marcadores ubicados en los extremos del controlador. La anchura (diámetro aproximado 6 cm) era adecuada para que **la cámara no tuviera problemas al reconocer los marcadores** a la vez que **era cómodo** para el jugador a la hora de sujetarlo con una **única mano**. El uso de la goma eva y el churro de piscina además era una forma de crear un controlador **barato**, **sencillo** y **difícil de romper**.

Una vez realizado el último prototipo propuesto se tenía bastante conocimiento sobre el dominio del problema y se empezó con la creación de un prototipo del que se pudiera realizar un video que pudiera ser enviado al cliente (tutor). El video enviado debía servir para **mostrar las capacidades** del prototipo y **verificar la viabilidad del proyecto** además serviría la **obtención de un feedback** del cliente (tutor) que pudiera servir para implementar la solución final.

Salvo por unas pequeñas modificaciones el programa de seguimiento no se modificó ya que había demostrado un funcionamiento bastante correcto, por lo que el siguiente prototipo a mejorar fue el del juego, al cual se había invertido menos tiempo.

Para la creación del prototipo del juego a entregar al tutor se empezó un proyecto nuevo en Unity, para su realización se aprovecharon las funciones de recepción de datos y normalización del prototipo_2. Las **coordenadas recibidas** de los 2 marcadores se debían procesar para **extraer la rotación y traslación de un único objeto**, en este caso un cilindro creado en Unity.

Aunque la **posición** de un objeto a partir de 2 coordenadas era muy fácil de calcular (punto medio entre ambos puntos) la **rotación** del objeto dio bastantes problemas, Unity usaba Quaternions para manejar la rotación de los objetos, sin embargo conceptualmente me resultaba más fácil resolver el problema mediante ángulos de Euler, por lo que se intentó hacer los cálculos mediante los ángulos Euler y luego traducirlos a Quaternion con las funciones de Unity. El proceso dio bastantes problemas y produciendo varios errores en las pruebas (e.g el objeto rotaba bien en algunas posiciones, en otras se invertía la rotación real). Al final se escogió una solución más sencilla utilizando una función de Unity que permitía calcular la rotación del objeto a partir de 2 vectores.

Una vez implementado el **movimiento y rotación** del cilindro se procedió a buscar una manera de probar si el seguimiento del controlador permitía la suficiente precisión y un tiempo de respuesta adecuado. Para realizar esta tarea se creó un **robot** que **disparara esferas** al jugador (el cual estaba situado delante del cilindro), para **añadir dificultad** se especificó que el **robot se moviera aleatoriamente** dentro de unos límites (para no salir de la vista del jugador), además el robot dispararía de una forma aparentemente aleatoria, de forma que al jugador le fuera más difícil predecir el momento disparo. Este tipo de mecánica se implementó ya que provocaba que el jugador tuviera que hacer usos de sus reflejos y por lo tanto que los movimientos del controlador serían rápidos (de esta forma se podían probar su funcionamiento para los casos más difíciles). Las pruebas realizadas ofrecieron unos buenos resultados, aunque la velocidad de respuesta no era la óptima el **jugador era capaz de parar** los disparos del robot.

Para mejorar el prototipo del juego y ofrecer al usuario un feedback se procedió a:

- La implementación de un **contador de vida** para el jugador, el cual se vería **reducido** al ser **impactado** por las esferas lanzadas por el robot. El contador de vida se vería reflejado en forma de **barra de vida** en la parte inferior izquierda.

- La **mejora gráfica**: creación de un suelo con relieve, mejorar la espada láser diferenciando el **mango** de la **hoja**, aplicación de un cielo (**skybox**) y aplicación de **texturas y colores** a los elementos del juego.

- Un **contador de esferas paradas** con el sable láser y un **contador de golpes recibidos**, mostrados en forma de **texto** en la parte inferior izquierda

- Una opción de **reiniciar el juego** si el contador de vida del jugador llegaba a 0, además de mostrarse en forma de texto el **tiempo de juego** de esa partida.

- **Información** de los **paquetes UDP recibidos** ubicada en la parte superior izquierda (este elemento es parte del código de [75] y fue utilizado para la realización de pruebas)

- **Creación de sonidos** mediante un programa online gratuito [76], se utilizaron **sonidos** para el **disparo del robot** y para la **parada** mediante el sable láser.

- En caso de que **se perdiera la visibilidad de un marcador** el mensaje contendría la **última coordenada** que fuera **visible** (esto permite hacer nuevos movimientos fijando un marcador en un punto y ocultándolo con el otro por ejemplo)

Una vez creado el prototipo y habiendo realizado pruebas para comprobar su consistencia se realizó un video con el programa gratuito [95], el cual fue enviado al tutor en el **informe de seguimiento** el cual contenía con una breve explicación de su funcionamiento, fallos a solucionar y posibles ampliaciones. A continuación se mostrarán algunas imágenes del video enviado

Ilustración 33 Video entregado en informe de seguimiento (I)

Tras la **recepción del feedback** del tutor se procedió a la refinación del prototipo, para ello se **reescribió el código** del prototipo para **optimizarlo** y **corregir el mayor número de errores posibles**, la actualización del prototipo trajo varios cambios:

- **Se implementó un contador** en el juego que calculaba cuantos **mensajes** llegaban **por segundo** y mostraba por la consola la media al finalizar la partida.

- Al reescribir el código del programa de seguimiento **se solucionaron varios errores**, entre ellos el que provocaba la interrupción del programa al minuto y medio (estaba provocado por una inicialización nula).

- **Se modificó el puntero** que indicaba la posición de los objetos, haciendo que destacara más.

- **Se mejoró el filtro de ruido** (reduce la posibilidad de error al haber otros colores similares al de los de los marcadores en el entorno).

- **Se unificó** en una matriz **los datos de la imagen binaria** (color blanco si el color ha sido encontrado y negro en caso contrario) de ambos marcadores para que pudieran ser mostrados en una única ventana.

- **Se eliminó** la necesidad del uso de la **librería OpenGL** reduciendo el número de librerías necesarias a 2, **OpenNI** y **OpenCV** (las funciones de OpenGL fueron sustituidas por funciones de la librería OpenCV).

- **Se especificó el tamaño y posición de las 2 ventanas y de la consola** para que nada más empezar el programa estuvieran ordenadas evitando solapamientos, facilitando de esta manera la ejecución de pruebas.

Uno de los objetivos más importante es que el programa de seguimiento **enviara las coordenadas al juego lo más rápido posible** ya que el tiempo de respuesta era muy importante para causar una buena sensación de control al jugador. La cámara Kinect era capaz de enviar como **máximo 30 fotogramas por segundo** al programa de seguimiento, por lo que estaba **limitado** el máximo de mensajes por segundo que podíamos recibir en Unity.

Para mejorar la rapidez se **optimizó el código del programa de seguimiento,** eliminando distintas funciones y comprobando los resultados que arrojaba el juego añadiendo un contador en el juego el cual calculaba la media de mensajes por segundo recibidos. En la siguiente tabla se muestran los resultados de la optimización más relevantes.

Cambios realizados en el programa de seguimiento	Mensajes por segundo recibidos (Unity)
Ningún cambio	17
Eliminando impresiones por pantalla	25.6
Evitando mostrar los datos en las ventanas	30 (aproximadamente)

De estas pruebas se pudieron extraer las siguientes conclusiones

- La **impresión en la consola** (mediante la función cout) de datos de control **retrasaba** bastante **la actualización de la posición y rotación** del controlador. Debido a que era una diferencia considerable de tiempos y la información ofrecida no era demasiado relevante se eliminaron todas aquellas impresiones de pantalla que no fueran necesarias, dejando únicamente aquellas que no estuvieran en un bucle.

- **La visualización de los datos en la pantalla** la cual sí que era necesaria para hacer las pruebas y comprender los posibles errores **no se eliminó**, sin embargo debido a la mejora que suponía evitar mostrar los datos se planteó crear en el futuro una **opción para habilitar o deshabilitar la información** de estas.

Lo más sorprendente de todo era que aplicando todas las mejoras al programa de seguimiento se **era capaz de alcanzar** en el juego **un numero de mensajes por segundo igual a la tasa de refresco máxima de la cámara** (máximo 30 fps). Por lo tanto usar OpenCV para procesar las imágenes y UDP para enviar los datos demostró ser una buena elección.

La siguiente fase del proyecto constaba de **actualizar el prototipo de juego permitiendo el uso de realidad virtual.** De esta manera el jugador podría rotar su cabeza para visualizar el entorno en el que se encontraba percibiendo de esta manera mejor los elementos de la escena (e.g esfera disparada por el robot) y provocando una mayor inmersión en el juego.

Para la implementación de la solución se utilizó el HMD (head mounted display) **Oculus Rift DK1** prestado por el departamento de informática de la Universidad Carlos III.

Una vez con el HMD en posesión se procedió a la lectura de la documentación, además de probar el dispositivo en diversas aplicaciones para comprender sus capacidades. Tras varias pruebas se descubrió que **Unity 5 soportaba Oculus Rift de forma directa** y tras mirar la

documentación de Unity la implementación de un sistema en el que el jugador pudiera rotar la cámara para visualizar el entorno con el movimiento del casco se realizó rápidamente.

Tal y como mostraba la documentación de Oculus, la versión DK1 inducía al mareo bastante fácilmente, especialmente al usarlas en largos periodos de tiempo o en aplicaciones en las que el personaje se moviera de forma predeterminada sin control del usuario. Se percibió también el efecto "screen door" (sensación de mirar a través de una rejilla) provocado por la resolución de la pantalla de las gafas.

A pesar de los problemas al probar las gafas en el juego la sensación de inmersión era bastante grande y **la presencia de elementos fijos** como el cielo **disminuía la sensación de mareo** (este consejo fue extraído de la guía de buenas prácticas de Oculus [96]).

La siguiente fase fue el desarrollo de una nueva versión del juego.

Utilizando el **feedback del informe de seguimiento** se plantearon varias ideas. Para visualizar las posibles mejoras y mecánicas de la nueva versión del juego se hizo uso del prototipado desechable utilizando bocetos. En la siguiente imagen se muestra un boceto de una de las ideas planteadas.

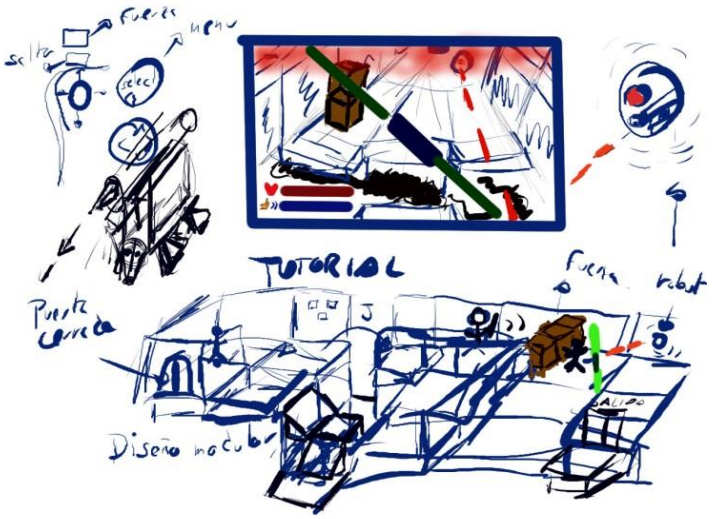


Ilustración 34 Bocetado de juego de nivel lineal

La idea era que el jugador se moviera por una nave o complejo industrial de forma más o menos lineal de manera que las diferentes mecánicas del juego se le fueran mostrando poco a poco a modo de tutorial. **Se acabó descartando** debido al poco tiempo de juego que se podría ofrecer. Además el crear un nivel lineal ofrecía al jugador una **experiencia más restringida** haciendo que fuera difícil plantear su rejugabilidad. A pesar de ello parte de las mecánicas pensadas se transmitieron a la solución final y ayudaron a pensar futuras ampliaciones del proyecto.

El feedback recibido por el tutor en el primer informe de seguimiento requería una **mejora a nivel gráfico** del juego. Para ello se empezó a buscar información sobre cómo crear un **shader** para representar un **halo de energía** que rodeara a la hoja del sable láser, tras varios intentos buscando alternativas y viendo la complejidad del problema se recurrió a la utilización del código del tutorial [97] para hacer tanto el efecto de halo como el efecto de la hoja.

Otro factor a mejorar era la **empuñadura del sable láser**, para ello se hizo uso del programa **Blender**, para el modelado se usó como base el tutorial [98] que también sirvió para introducirse al programa y los tipos de técnicas que se podían utilizar para modelar un objeto en 3D.

Otro de los consejos recibidos fue hacer uso de un **sable láser de doble hoja** para que se asemejara más a la forma que tenía el jugador de agarrar el controlador físico de gomaespuma (se agarra por el centro para dejar visibles los marcadores). Para ello se modeló la empuñadura de un sable láser de doble hoja.

Sin embargo al hacer las pruebas se comprobó que utilizar un modelo de una empuñadura de un sable láser de doble hoja en el juego hacia que este **ocupara gran parte de la vista del jugador**, ocupando la zona central y dejando las hojas demasiado alejadas. Debido a que gran parte de la funcionalidad recaía en las hojas, y la empuñadura era mucho menos relevante, se decidió intentar reducirlo de tamaño. A pesar de ello seguía ocupando demasiado espacio por lo que se descartó el modelo y se creó otra empuñadura para un sable de una única hoja. A continuación se puede ver algunas imágenes del proceso de pruebas y modelado

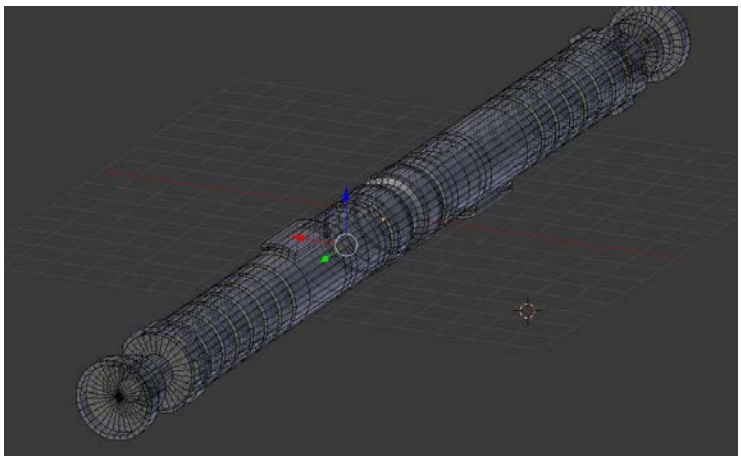


Ilustración 35 Modelado 3D en Blender de empuñadura de un sable láser de doble hoja

Ilustración 36 Pruebas en Unity realizadas a empuñadura de doble hoja con material metálico

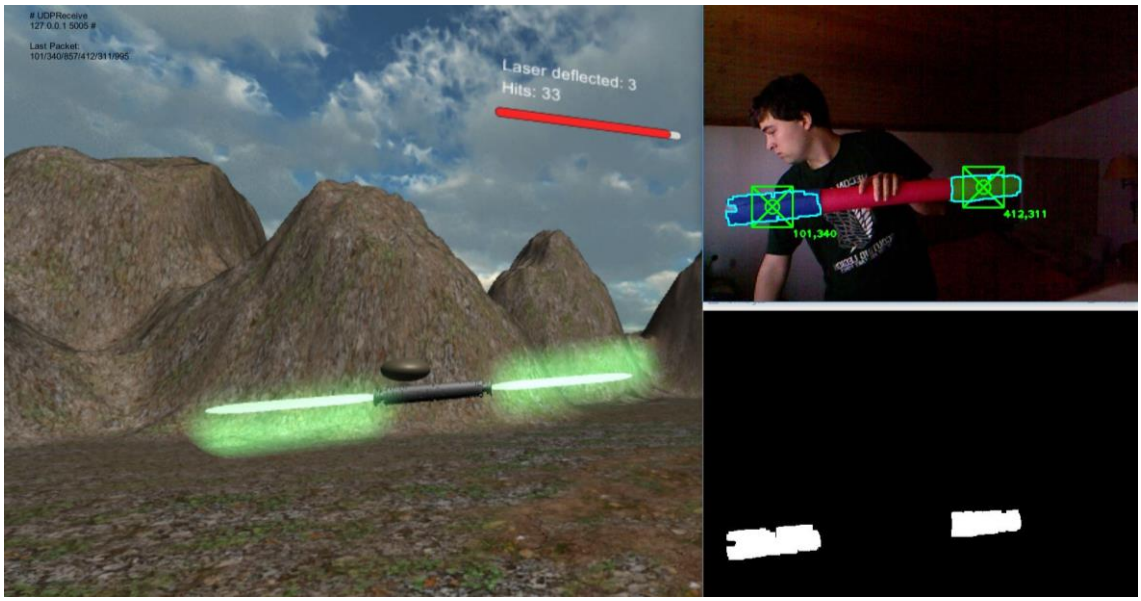


Ilustración 37 Pruebas realizadas en Unity de jugabilidad con sable de doble hoja

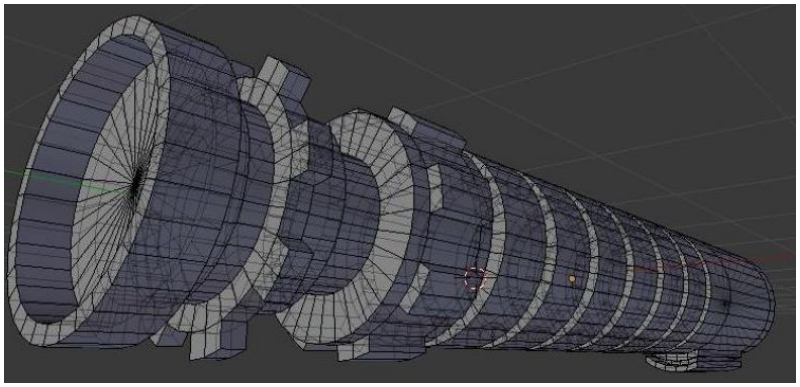


Ilustración 38 Modelado 3D en Blender de empuñadura de un sable láser de una hoja

Para darle un aspecto metálico se utilizó un material creado en Unity de color gris con propiedades reflectantes metálicas.

Otra mejora grafica que se podía realizar mediante Blender **era la del robot**. Utilizando los conocimientos adquiridos en el modelado de la empuñadura del sable láser se realizó un modelo 3D del robot enemigo. El material usado para simular el efecto metálico era similar al de la empuñadura del sable láser con el color modificado.

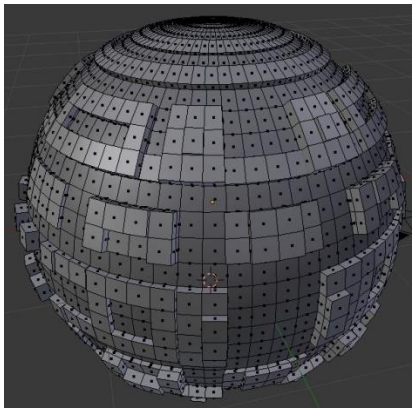


Ilustración 39 Modelo 3D robot en Blender

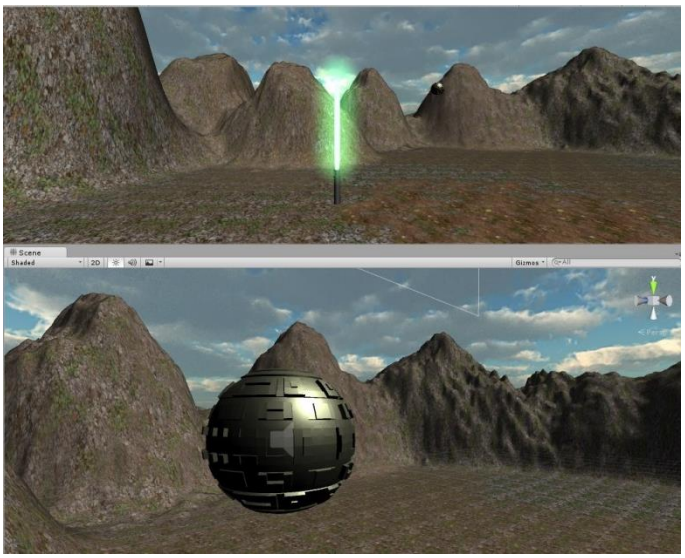


Ilustración 40 Pruebas en Unity de material metálico en robot

La siguiente mejora era dotar a las esferas que simulaban ser proyectiles de un aspecto de **disparo láser**, para ello se utilizó un área de luz ya implementa en Unity que creaba un halo de luz esférico alrededor de un punto. Se escogió darle **color rojo** para que el jugador pudiera **diferenciarlo fácilmente** del resto del entorno. El interior de la esfera, la parte sólida utiliza el mismo shader obtenido de [97] utilizado en la parte solida de la hoja del sable láser.

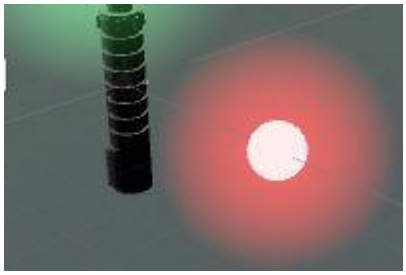


Ilustración 41 Pruebas en Unity de proyectil láser con halo rojo

Otra de las posibles mejoras planteadas tras el informe de seguimiento (I) era **dotar al personaje de movimiento** mediante un **mando Xbox 360 inalámbrico** del cual me hallaba en posesión.

El mando Xbox 360 cuenta con compatibilidad directa con Unity por lo que utilizando los datos del **joystick izquierdo** se creó una función que moviera al personaje en el juego. Ya que el **sable láser debía acompañar al jugador** se agregó a la función de posicionamiento del sable láser la posición actual del personaje. De esta manera se sumaba la posición del jugador con la del sable láser teniendo como efecto que **este acompañara al personaje** fuera donde fuera.

Se decidió que ya que el jugador no podía acceder al joystick derecho únicamente con la mano izquierda, **la rotación sería controlada mediante la rotación del casco Oculus Rift**. La implementación de la rotación del personaje fue más complicada de lo esperado debido a problemas con la forma de tratar los datos de la cámara VR de Unity haciendo que el personaje rotara en todos los ejes a pesar de especificar que solo rotara en función del eje Y. Tras varios intentos se descubrió que la solución era agregar un objeto vacío como padre de la cámara.

Otra de las funcionalidades que se probaron fue la de que el sable láser rotará también en función de la rotación del personaje. Sin embargo se encontró molesto el que parte de la vista estuviera siempre ocupada por el sable láser, de forma que **el seguimiento de la rotación del jugador por parte del sable láser fue descartado**.

Se **mejoró el sistema de movimiento del sable láser**, haciéndolo más preciso, para ello se modificaron las coordenadas enviadas por el programa de seguimiento de forma que las **coordenadas x, y, z** de cada marcador **fueran enviadas en formato de milímetros**.

Dentro del juego se dividió el movimiento del sable láser en **posicionamiento** y **rotación**, las mejoras introducidas garantizaban una **respuesta más acorde con los movimientos del jugador** lo cual también provocaba que los fallos a la hora de posicionar los marcadores se notaran más.

Otro factor a mejorar era que el proyectil rebotara contra el sable láser, para ello se implementó un **material físico** en Unity de baja fricción que rebotará al colisionar con otros objetos. Se modificaron las funciones de colisión del proyectil haciendo que este **rebotara únicamente en la hoja del sable láser** destruyéndose en los demás casos.

La siguiente fase consistía en dotar de **dinamismo y realismo al mapa** para dar al jugador una **mayor inmersión**. Para ello se pensó en las batallas navales organizadas en el coliseo romano y posibles mecánicas que pudieran tener sentido en este. Tras varias ideas se pensó que sería interesante contar con un **coliseo** en el que el centro estuviera cubierto por agua de forma que **si el que el personaje cayera al agua este moriría**.

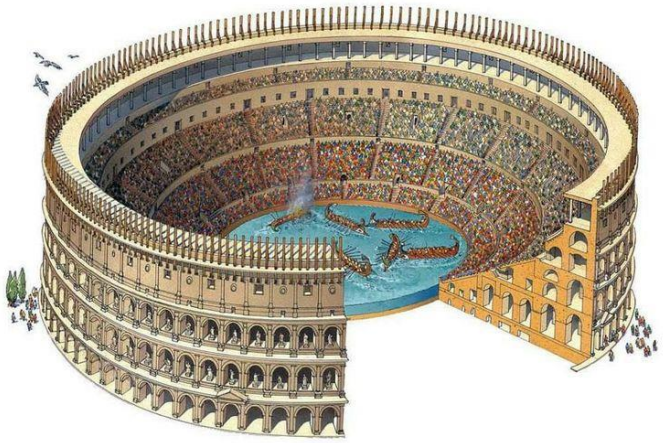


Ilustración 42 Representación de una batalla naval celebrada en el coliseo romano

El suelo donde lucharía el jugador contra el robot estaría **compuesto** por una **serie de placas creadas al inicio del juego**. En intervalos aleatorios se seleccionarían varias placas para caer atravesando el agua. El **número de placas seleccionadas se incrementaría** a medida que el tiempo del juego pasara **provocando un reto cada vez mayor** al jugador ya que tendría que seguir concentrado en parar los disparos del robot además de moverse para evitar caer al agua.

Para la creación del modelo 3D del coliseo se utilizó Blender. Aplicando las lecciones aprendidas en el modelado del sable láser se creó un coliseo con gradas y varios elementos como palcos o puertas que ayudaran al jugador a **orientarse** dentro del juego utilizándolos como **puntos de referencia** (recordar que el uso de puntos fijos evitaba el mareo del jugador [96]).

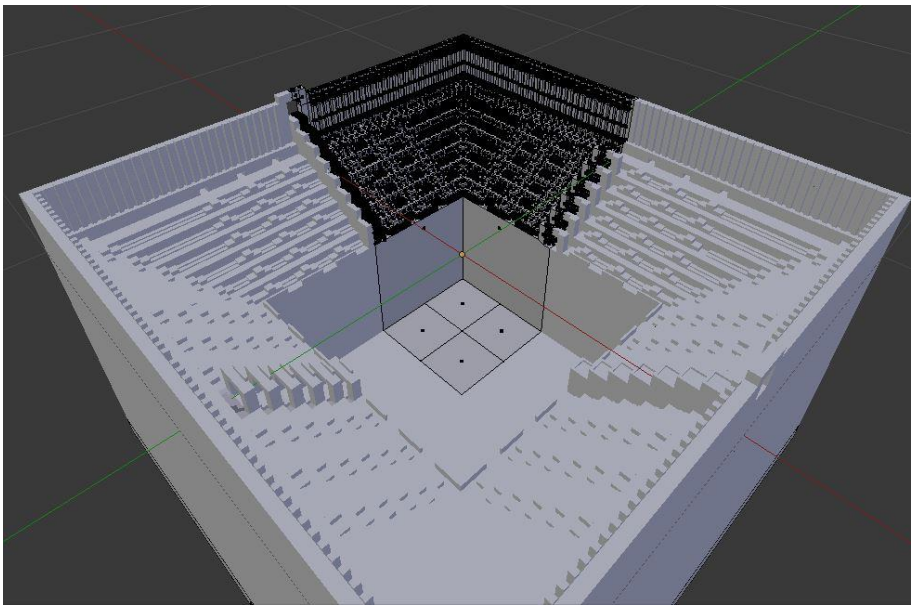


Ilustración 43 Modelado 3D del coliseo en Blender

Para el agua se utilizó un recurso importado de Unity posicionándolo en el centro de la parte inferior del estadio.

Para el suelo formado de placas se utilizó cubos de Unity modificados con el tamaño deseado. Se implementaron **dos tipos de placas** de diferente color y se especificó que **al iniciar el juego** se instanciaran con un patrón de un tablero de ajedrez para que el jugador **pudiera diferenciar una placa de sus contiguas**.

El comportamiento de **caída y subida** fue implementado en las propias placas las cuales al activarse dicho método descenderían y ascenderían mediante el uso de temporizadores. Para proporcionar al jugador un feedback cuando la placa **esté cerca de caer** se cambiará el color de la placa a uno **amarrillo** y cuando **esté a punto de caer** a **rojo.** Al volver a su posición retornara el color original a la placa. En ese proceso también se proporcionara un **feedback en forma de audio** para cada estado.

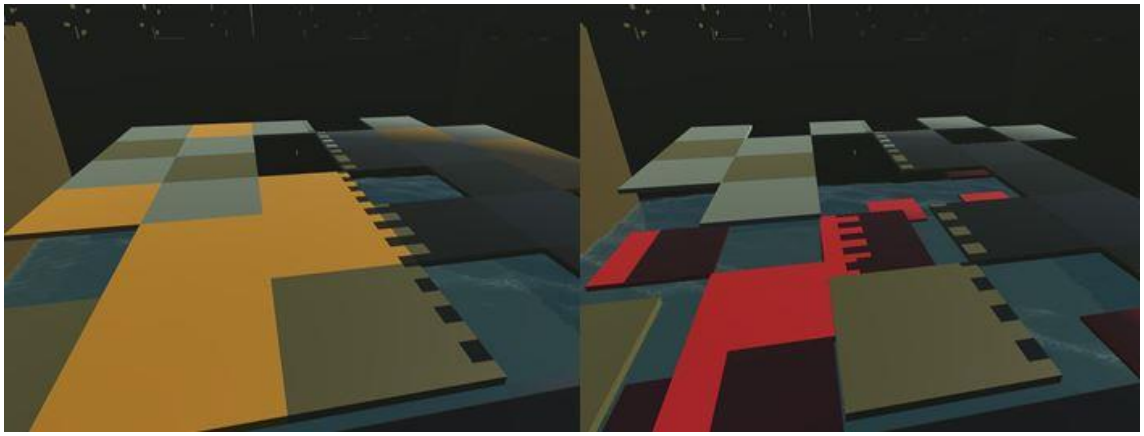


Ilustración 44 Proceso de caída de placas

Para dotar al jugador de **mayor movilidad** se dotó el gatillo del mando o a la barra espaciadora en el teclado la función de **salto** del personaje. Se otorgó también la posibilidad de realizar un

doble salto tras un corto periodo de tiempo. Gracias a la implementación del salto el jugador ya podía moverse con mucha **mayor fluidez** entre las placas pudiendo evitar caer al agua por ejemplo saltando de una placa que estuviera cayendo al agua a una que estuviera subiendo.

La inclusión de placas que subieran y bajaran permitía provocar también al jugador una **mayor inmersión,** por ejemplo pudiéndose asomar a los huecos que dejaban las placas al caer para contemplar el agua.

Para dotar de dinamismo al robot se creó una función que **persiguiera al jugador** a lo largo del espacio 3D intentando siempre ubicarse en **un rango desde el cual tendría permitido disparar**.

El rango siempre estaría orientado a la orientación original del personaje, ignorando las demás rotaciones que el personaje pudiera hacer con el casco, de esa forma el robot siempre dispararía al jugador estando **el sable láser ubicado entre ambos**.

Ya que la cámara Kinect estaría **situada en un lugar fijo** esto permitiría que el jugador cada vez que quisiera interceptar los proyectiles **tuviera que rotar hacia la cámara Kinect** la cual podría mediante el programa de seguimiento mandar las coordenadas de los dos marcadores. Para facilitar este proceso y que la cámara no tuviera que estar siempre en un mismo lugar se implementó la opción de **resetear la rotación del personaje a la inicial.** De esta forma el propio jugador podría **recentrar al personaje** para recibir los proyectiles mirando hacia la cámara Kinect, para ello se habilitó el botón del joystick izquierdo del mando de Xbox 360 o la tecla “c” del teclado. El habilitar teclas tanto en el teclado como el mando permitía la existencia de un ayudante que pudiera asistir al jugador a orientarse hacia la cámara al inicio de la partida y recentrar la posición desde el teclado

Dado que también se quería conseguir que el jugador pudiera acostumbrarse a las gafas y no iniciar el juego con el robot disparándole se habilito el botón “select” del mando Xbox (botón situado arriba del disparador izquierdo) o la tecla “h” para que pudiera **activar o desactivar al robot** haciéndolo aparecer y desaparecer del escenario. Se implementó que **por defecto el robot no estuviera activado** y que fuera el jugador el que lo activará cuando estuviera preparado.

Siguiendo el feedback del tutor respecto a las espadas láser de doble hoja se habilito el botón “lb” del mando Xbox (botón situado arriba del disparador izquierdo) o la tecla “l” para **poder encender una segunda hoja** en el sable láser junto con la otra.

El sable del jugador se **inicializaría desconectado**, un click **encendería la hoja superior** del sable láser, otro click **encendería** también **la hoja inferior** del sable láser, otro click volvería al estado original **apagando ambas hojas**. Para añadir un feedback auditivo al usuario se dotó de sonidos de encendido y apagado a cada hoja. Ya que ya se había comprobado que utilizar un mango de doble hoja era una mala opción se mantuvo el modelo de la empuñadura del sable láser individual.

Para dotar de mayor realismo al juego se usaron **sitemas de particulas** incluidos en Unity para representar la colisión del proyectil láser en una superficie, en caso de colisionar con el suelo también aplicaría una textura de rotura que **perduraría 3 segundos** (de está manera no hay que preocuparse que el juego se ralentize por la presencia de demasiadas texturas de rotura acumuladas).

Ya que mediante las gafas de realidad virtual no era conveniente poner un texto de derrota que acompañara la visión del jugador (ya que produce incomodidades y mareos) se añadió el

texto informativo de derrota en el cielo junto a las estadisticas de la partida (barra de vida, golpes recibidos y proyectiles deflactados).

Situado en el cielo de forma que el jugador pudiera tener una visión agradable del texto se ubicó una superficie a la cual sería transportado el jugador al morir (por caida al agua o por proyectiles recibidos), desde la cual el jugador podría visualizar sus estadisticas. Con el objetivo de proporcionar una **mayor sensación de inmersión** al jugador se utilizó un material cristalino para que el jugador pudiera ver todo el escenario bajo sus pies.

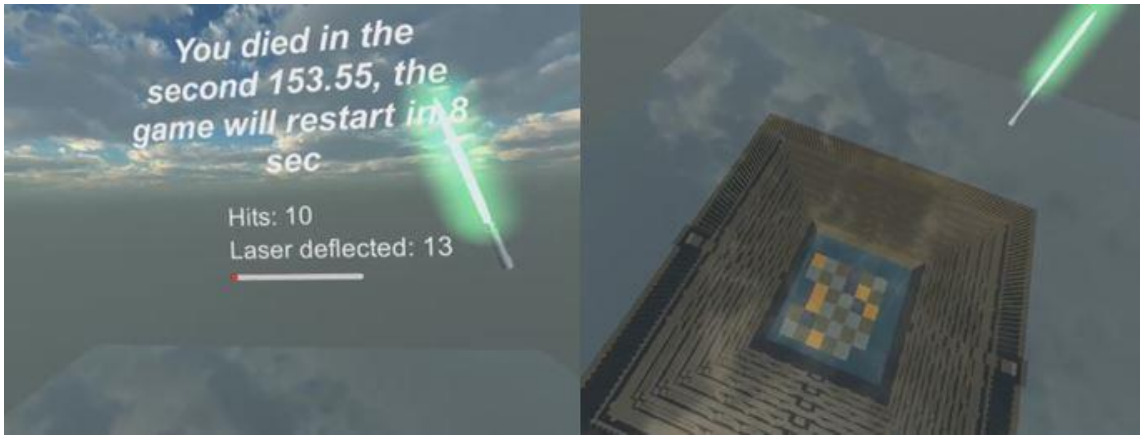


Ilustración 45 Vistas en la superficie cristalina tras la muerte del personaje

Para evitar que el jugador tuviera que presionar ningún botón para resetear el juego al morir se estableció un **contador de 8 segundos** tras el cual el **juego sería reiniciado** automaticamente.

Para otorgar una opción al jugador de **ganar el juego,** no solo de ser derrotado, se implementó que el jugador pudiera conseguir la **victoria** en caso de que **consiguiera deflectar uno de los proyectiles al robot**, en cuyo caso se transportaría al personaje a la plataforma cristalina mostrandole un texto informandole de su victoria. Para añadir más feedback al usuario se usaron las particulas de Unity para **instanciar fuegos artificiales** alrededor de la plataforma. Al cabo de **10 segundos** el juego se reinciaría automaticamente.

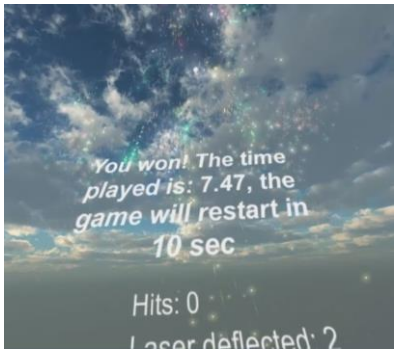


Ilustración 46 Vistas en la superficie cristalina tras la victoria

Se proporcionó al juego de **varios efectos de audio**, para aumentar el feedback e inmersión del usuario. Se añadió **sonido ambiental** al escenario utilizando sonidos de **viento y agua**. Para provocar una sensación de vertigo al estar en la superficie cristalina se especificó que el **volumen máximo** alcanzado por el sonido del viento debería ser en la superficie cristalina, **disminuyendose** a medida que el jugador se **alejase de ella**. De esta forma el jugador al estar

en el coliseo escucharía el sonido del viento de una forma leve, como una brisa, ya que se encuentra resguardado por las paredes del coliseo ese debería ser el efecto real. Por el contrario al estar en una superficie en el cielo sin resguardo el sonido viento debería ser mucho más violento.

Para dotar a las colisiones de disparos de más información para el jugador se añadió la reproducción de audio cada vez que colisionaran y **dependiendo del superfice** en la que impactara se reproduciría un tipo de audio u otro.

Ya que las colisiones de los proyectiles en muchos casos **destruian el objeto** no era posible encargar de la reproducción del audio a los propios proyectiles. Para solucionar este problema se creó un **objeto invisible auxiliar** el cual sería instanciado cuando se quisiera reproducir un archivo de audio. En dicho objeto se encontrarían varios archivos de audio, dependiendo de los parametros de instanciación reproduciría aquel que correspondiera (e.g golpe contra el suelo, golpe contra sable láser). El **objeto auxiliar será destruido a la finalización del audio**, evitando la presencia de demasiados objetos auxiliares de audio al mismo tiempo.

Para obtener los sonidos de **salto**, **aterrizaje** y **herida** se utilizo la herramienta gratuita Audacity con la que se grabó y editó el audio (e.g se aplicaron filtros de ruido para que el sonido fuese de mejor calidad).

Para evitar la sensación de repetición se hace uso de varios sonidos de cada tipo, los cuales serán seleccionados de forma aleatoria, **aumentando** de esta manera **las combinaciones posibles** (e.g al recibir un impacto se seleccionará aleatoriamente uno de los 5 audios de impacto láser y uno de los 5 audios de recepción de daño creando 25 diferentes combinaciones).

Una vez finalizado el juego se comprobó la cantidad de FPS a la que ejecutaba el juego. Tal y como se especificaba en los requisitos **60 frames por segundo sería el resultado óptimo**, esto es debido a que el casco Oculus Rift DK1 opera a **60 Hz** pudiendo presentar un máximo de 60 ímagenes por segundo al jugador. El resultado fue satisfactorio, el juego operaba entre 63 y 66 FPS. Se podría intentar hacer alguna optimización para que la tasa de refresco fuera más fluida pero los cambios no sería visibles con la versión DK1 de Oculus.

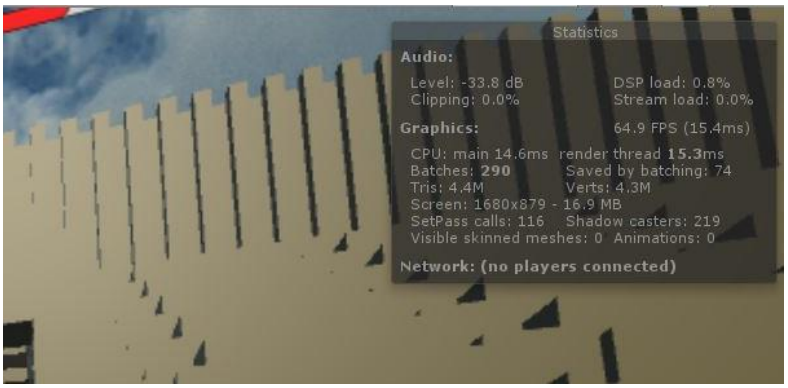


Ilustración 47 Imagen con las estadísticas en la ejecución del juego

Explicación de la solución

Donde se explica de forma más detallada el funcionamiento de ambos programas.

Explicación programa de seguimiento

Al iniciar el programa se crean las dos ventanas auxiliares que nos servirán para visualizar las pruebas, un video a color y un video de la imagen binaria procesada en blanco negro, junto con la consola se **escalan** y se **posicionan** de forma que sea **sencillo visualizar la información**.

Mediante OpenCV se pueden percibir **operaciones del mouse** cuando está ubicado en una ventana, esto es realizado mediante la función **callback**. Una función **callback** es una función que se llama al producirse **eventos** (por ejemplo click derecho) y que provoca la actuación de un **handler**, el cual en nuestro caso es el que registra los pixeles seleccionados en la ventana de color los cuales sirven para hacer la selección de color a seguir posteriormente. Para más información consultar el capítulo 1 “Loading, displaying, and saving images” de [99]o [91].

A continuación inicializamos OpenNI y abrimos el dispositivo Kinect. OpenNI hace uso del objeto VideoStream para acceder a los datos de la cámara Kinect (color, IR, y sensores de profundidad), para más información véase introducción de [90]. Una vez abierto el dispositivo se solicita que cree el **stream de profundidad** y se configuran las **características** de este (30 fps, resolución 640x480, PIXEL_FORMAT_DEPTH_1_MM (formato de profundidad usual, 1mm de precisión)). A continuación realizamos el mismo proceso para el **sensor de color** modificando el formato de pixel a PIXEL_FORMAT_RGB888 (es posible usar este formato tanto para el stream de color como para el de IR, genera un bit map de 24 bit). Para más información sobre los diferentes formatos de pixel mirar capítulo 2 “Accessing video streams (depth/IR/RGB) and configuring them” de [90]).

Cuando tanto el stream de color y de profundidad están disponibles es posible que las imágenes enviadas por ambos **no se hayan obtenido al mismo tiempo**, para solucionarlo activamos la sincronización mediante **FrameSync**, de modo que la imagen a color y el mapa de profundidad estén separados por el menor tiempo posible.

Kinect posee dos cámaras, una que provee de imágenes a color (cámara RGB) y otra de la que se extrae el mapa de profundidad (cámara infrarroja), ambas están separadas y por lo tanto **las imágenes producidas no se corresponden entre sí.** Sin embargo al conocer la distancia que separa a las cámaras es posible realizar varios cálculos matemáticos para **alinear ambas imágenes** haciendo que los pixeles de ambas se correspondan entre sí. Este proceso es conocido como **Registration**, en el caso de Kinect esos cálculos son realizados automáticamente por el hardware. Lo único necesario es activarlo mediante OpenNI con la función setImageRegistrationMode(IMAGE_TO_DEPTH_REGISTRATION_IMAGE), esto provoca también que se pierdan algunos valores del extremo del mapa de profundidad. En la siguiente imagen se puede ver el efecto producido.

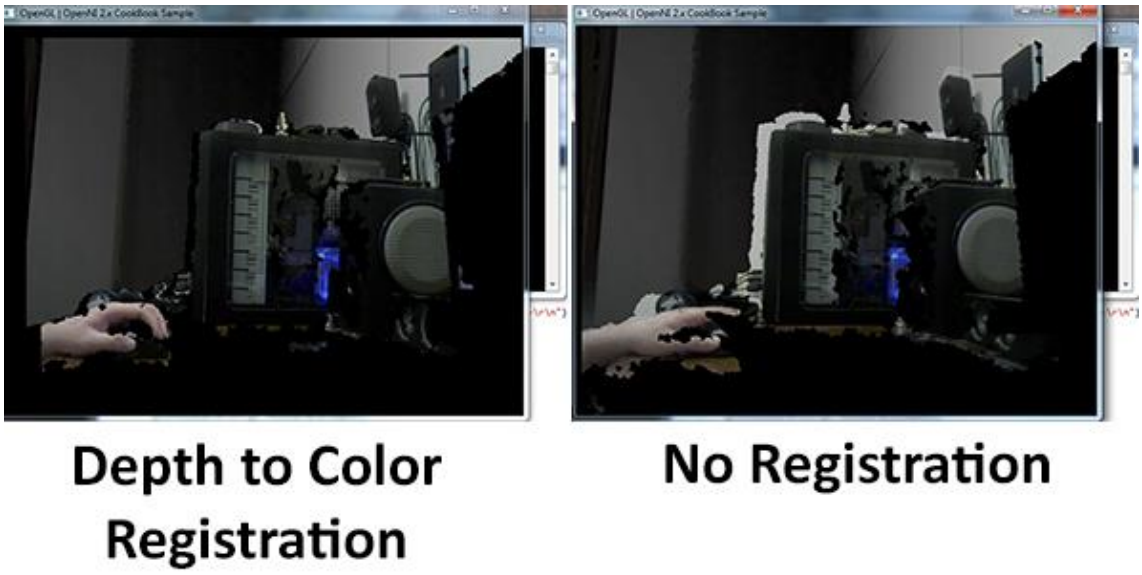


Ilustración 48 Diferencias entre alineado y no alineado de imagen de profundidad e imagen RGB extraído de [90]

```
	printf("Enabling Depth-Image frames sync\n");
	device.setDepthColorSyncEnabled(true);
	printf("Enabling Depth to Image mapping\n");
	device.setImageRegistrationMode(IMAGE_REGISTRATION_DEPTH_TO_COLOR);
```

Código 1 Alineación y sincronización de video a color con mapa de profundidad

Finalizada la configuración de la cámara se empieza el bucle principal en el que se **procesarán las imágenes** y obtendremos las **coordenadas de los marcadores**.

Declaramos las matrices donde guardaremos los datos y se comprueba si el estado del stream de color y profundidad es válido. El siguiente paso es esperar a que esté disponible una imagen (frame) del stream de profundidad, una vez esté disponible leemos el frame de profundidad y el de color. Los datos de ambos fotogramas son guardados en sus correspondientes matrices.

```
	if (depthSensor.isValid() && colorSensor.isValid())
	{
		VideoStream* streamPointer = &depthSensor;
		int streamReadyIndex;
		OpenNI::waitForAnyStream(&streamPointer, 1,&streamReadyIndex,
500);
		if (streamReadyIndex == 0) {
			VideoFrameRef depthFrame;
			depthSensor.readFrame(&depthFrame);
			VideoFrameRef colorFrame;
			colorSensor.readFrame(&colorFrame);
			depthcv.data = (uchar*)depthFrame.getData();
			colorMat.data = (uchar*)colorFrame.getData();
```

Código 2 Lectura y extracción de datos de los streams de color y profundidad

La **segmentación** en visión artificial es el proceso de **particionar una imagen en conjuntos de pixeles de características similares**. Segmentación puede ser usada para **distinguir un objeto de otro**, tal y como queremos hacer con los marcadores. En nuestro caso se quiere diferenciar los pixeles utilizando las propiedades del color.

La cámara de color de Kinect capta imágenes con formato RGB. El problema es que **RGB no es un buen método para hacer segmentación de colores**, para segmentación **es mejor usar HSV** (hue(tono), saturation(cantidad de color), value (intensidad de luz)) el cual se asemeja más al modo de organizar los colores que tiene el ser humano, pagina 54 [100]. El principal motivo de que HSV sobresalga ante RGB al hacer la segmentación es que HSV **separa la información del color de la intensidad de luz** (en RGB la intensidad de luz está incorporada en los valores de R, G y B). Aunque hay otros formatos que también hacen esa separación, HSV suele ser escogido ya que es sencillo transformar los valores RGB en HSV y viceversa.

OpenCV tiene una función que permite convertir el color llamada **cvtColor**, para transformar la imagen a HSV tendremos que utilizar el parámetro CV_BGR2HSV. No se utiliza CV_RGB2HSV ya que **OpenCV usa el formato BGR** (RGB invertido).

Para obtener los valores se utilizan las funciones **recordHSV_values** y **clickAndDrag_rectangle** las cuales están extraídas directamente del código de Kyle Hounslow [91] (con algunas modificaciones).

ClickAndDrag_rectangle es llamada cada vez que el ratón interactúa con la ventana que muestra el video a color. Si se presiona el botón izquierdo guarda las coordenadas x e y donde se hizo click la primera vez, según va **arrastrando el ratón** va guardando la posición actual del puntero del ratón **dibujando un rectángulo** en la ventana de video a color, que sirve como feedback al usuario.

RecordHSV_Values es una función que utiliza el valor del rectángulo para extraer los valores de los pixeles del fotograma (frame) y almacena los valores H, S y V en vectores distintos. Más tarde se recorren los vectores para encontrar **el valor mínimo y máximo de los tres**. Almacenaremos los datos en distintas variables dependiendo de qué marcador sea el rectángulo (click derecho en la pantalla de video para cambiar de marcador 1 a 2 o viceversa).

Estas dos funciones se intentaron realizar de manera distinta, cogiendo un único punto y con un valor prefijado establecer un mínimo y un máximo, sin embargo el resultado no fue muy bueno y se acabó descartando. Para más información sobre funciones que interactúen con el ratón consultar Tema 1 sección “Loading, displaying, and saving images” de [99], también es explicado en el video [74]. Para ahorrar tiempo en las pruebas se inicializaron los valores HSV mínimos y máximos con los valores HSV de azul y verde (color de marcadores).

Una vez se tienen los valores **HSV mínimos y máximos** se puede aplicar la función OpenCV **inRange** de la cual se extrae una **imagen binaria** que da el valor máximo (255) al elemento de la matriz si se encuentra entre ese máximo y mínimo mientras que para el resto de valores se da el valor mínimo (0). Representado en una imagen el color blanco correspondería con el blanco para el máximo valor y negro para el mínimo. En el tema 6 sección “Iris identification, how is it done?” de [101] utilizan esta función de segmentación para localizar la pupila de un ojo humano (la pupila es la región más oscura en las imágenes NIR (near infrared imaging)).

```
cvtColor(colorMat, HSV, COLOR_BGR2HSV);
recordHSV_Values(colorMat,HSV);
inRange(HSV, Scalar(H_MIN, S_MIN, V_MIN), Scalar(H_MAX, S_MAX, V_MAX),
binaryImage);
inRange(HSV, Scalar(H_MIN2, S_MIN2, V_MIN2), Scalar(H_MAX2, S_MAX2,
V_MAX2), binaryImage2);
```

Código 3 Conversión de color y procesamiento de imagen binaria

Una vez se tienen las imágenes binarias es necesario **aplicar un filtro** a estas para evitar el ruido. En este caso se hará uso de las operaciones morfológicas **erode** y **dilate**. Uno de los componentes de estas operaciones es el **elemento estructurado** el cual es una agrupación de pixeles (normalmente son escogidas agrupaciones como la del cuadrado o circulo). Ese **elemento estructurado** contara con un pixel llamado **punto de anclaje** (anchor point) normalmente ubicado en el **centro de la agrupación**. En la siguiente imagen se muestra un ejemplo de elemento estructurado con forma de cuadrado (los pixeles sombreados) y su punto de anclaje (la x).

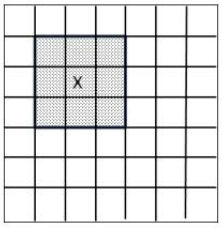


Ilustración 49 Ejemplo elemento estructurado extraída de [99]

Erode busca el menor valor entre los pixeles del elemento estructurado (en una imagen binaria el 0) y sustituye su punto de anclaje por dicho valor.

Dilate busca el mayor valor entre los pixeles del elemento estructurado (en una imagen binaria el 255) y sustituye su punto de anclaje por dicho valor.

Aplicando **erode** de manera repetida **eliminará agrupaciones de pixeles pequeñas** (ruido).

Tras la aplicación de **erode** las agrupaciones suficientemente grandes persistirán. Sin embargo **su tamaño se habrá reducido**, para solucionarlo aplicamos **dilate** lo cual hará las agrupaciones de pixeles existentes **más grandes** pudiéndose rellenar posibles agujeros en las agrupaciones.

Por defecto OpenCV usa un elemento estructurado de 3x3 pero podemos darle cualquier valor (cuanto más grande mayor será el efecto), el valor Point (-1,-1) en el campo anchor indica que se escoja como **punto de anclaje el centro**. Para más información consultar el tema 5 sección "Eroding and dilating images using morphological filters" de [99].

```
void applyFilter(Mat &binaryImage)
{
	Mat element(3, 3, CV_8U, cv::Scalar(1));
	erode(binaryImage, binaryImage, element, Point(-1, -1),2);
	dilate(binaryImage, binaryImage, element, Point(-1, -1), 5);
}
```

Código 4 Aplicación de filtros morfológicos

En este caso se aplica 2 veces la operación erode y 5 veces dilate. En las siguientes imagen se pueden ver los resultados de su aplicación.

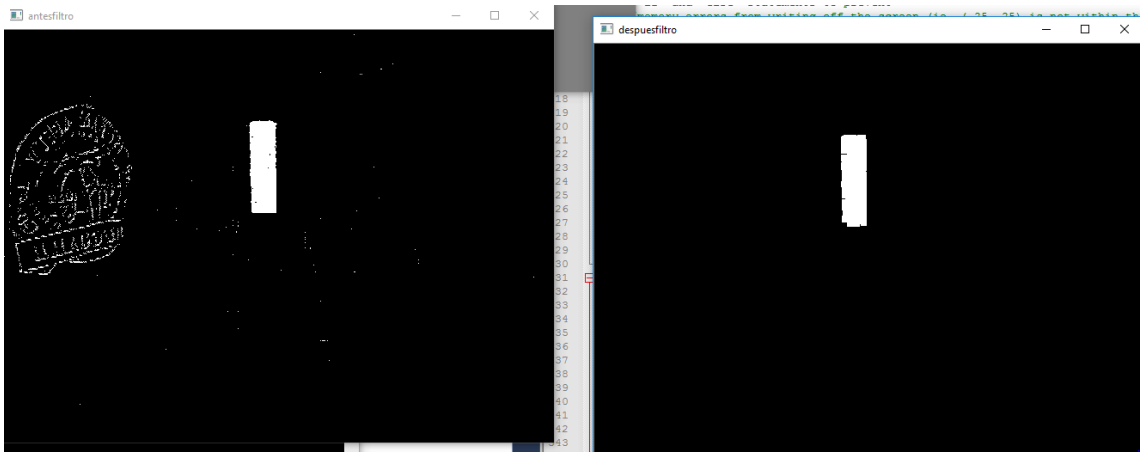


Ilustración 50 Reducción de ruido provocada mediante erode y dilate (izquierda, imagen binaria sin filtro, derecha, imagen con filtro aplicado)

Una vez se tienen las **imágenes binarias con los filtros aplicados** es el momento de **localizar** las **coordenadas de cada marcador**. Para ello declaramos un vector de vectores de puntos, un vector de puntos nos permite almacenar varios puntos que forman un contorno, pero puede que detectemos **varios conjuntos de contornos** de ahí la utilización de un vector de vectores de puntos.

Para obtener los contornos OpenCV posee la función **findCountours**. Esta función utiliza una imagen binaria para obtener un vector de vectores de puntos. Aunque tiene parámetros opcionales como hierarchy (vector de salida en el que se pueden almacenar contornos que estén dentro de otros contornos) no los utilizaremos.

Hay varios modos a la hora de extraer los contornos, en nuestro caso usaremos el parámetro CV_RETR_EXTERNAL que solo muestra los contornos exteriores, los únicos necesarios en nuestro caso.

El parámetro de aproximación de contornos elegido ha sido CV_CHAIN_APPROX_SIMPLE para almacenar menos puntos y acelerar el proceso del tratamiento de la imagen.

Para más información sobre el uso de findCountours consultar capítulo 7 sección “Extracting the components countours” de [99] o consultar la documentación de OpenCV [102].

```
bool trackObject(int &x, int &y, Mat binaryImage, Mat &cameraFeed) {
      int maxArea = 0;
      int maxCont = 0;
      bool objectFound = false;
      Mat binaryImageTemp;
      binaryImage.copyTo(binaryImageTemp);
      vector< vector<Point> > contours;
      findContours(binaryImageTemp,contours, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE);
```

Código 5 Obtención de contornos de imagen binaria

Una vez hemos **extraído los contornos** se inicia un bucle con tantas iteraciones como contornos hayamos encontrado, calculamos el área del contorno utilizando los **momentos** los cuales describen la forma del objeto, capitulo 7 sección “Computing components shape

descriptors" de [99]. Una vez **obtenemos el área** aplicamos un filtro para escoger el objeto de **mayor área** (el cual debería ser nuestro marcador).

En caso de que el filtro de erode y dilate no haya funcionado bien quedando contornos con un área muy pequeña (ruido) se restringirá a que el contorno detectado tenga un área mínima de 40x40 (esta idea fue extraída gracias al proyecto de Kyle Hounslow [91]).

Una vez obtenemos las coordenadas x e y del objeto, se dibuja en la imagen a color un **puntero** (el puntero es una composición de figuras simples, para más información sobre cómo usar figuras simples consultar [103])

Junto al puntero se añade el valor de las coordenadas (que representan la posición de la matriz en la que se encuentran, no la posición 3D final), además también **se dibuja el contorno más grande encontrado** para ayudar al usuario a localizar el objeto.

```
for (int i = 0; i < contours.size();i++) {
      Moments moment = moments(contours[i]);
      double area = moment.m00;
      if (area>minArea&&area>maxArea) {
            x = moment.m10 / area;
            y = moment.m01 / area;
            objectFound = true;
            maxArea = area;
            maxCont = i;
      }
}
if (objectFound == true) {
      drawObject(x, y, cameraFeed);
      drawContours(cameraFeed, contours, maxCont, yellow, 2);
}
return objectFound;
```

Código 6 Calculo del centro del objeto y dibujado de puntero y bordes

En caso de haber encontrado las coordenadas del objeto procederemos a **calcular la profundidad del pixel ubicado en dichas coordenadas**, para ello utilizaremos el **mapa de profundidad**. Mediante la función **convertDepthToWorld** extraeremos la profundidad en milímetros. En el capítulo 4 sección "Converting the depth unit to millimetre" de [90] hay un ejemplo que explica más en detalle su funcionamiento. Mediante este método también se extrae las coordenadas reales x e y en milímetros.

```
if (trackObject(xObj1, yObj1, binaryImage, colorMat)) {
      DepthPixel* pixel1 =
            (DepthPixel*)((char*)depthFrame.getData() +
                  (yObj1  *
                        depthFrame.getStrideInBytes()))
            + (xObj1);

      CoordinateConverter::convertDepthToWorld(
            depthSensor,
            (float)(xObj1),
            (float)(yObj1),
            (float)(*pixel1),
            &wXObj1, &wYObj1, &wZObj1);
}
```

Código 7 Obtención de coordenadas reales a mm mediante OpenNI

Antes de enviar las coordenadas x, y, z del color encontrado **aplicamos un filtro que evite grandes diferencias de profundidad**. Esto hará que posibles errores en la lectura sean sustituidos por el valor anterior (en caso de producirse tales fallos puntuales sin filtros el objeto se movería a una profundidad inusual y luego volvería a su posición normal). También se evita que la cámara envié mensajes si el objeto está ubicado a **menos de 0.8 m**, límite para la cámara Kinect V1 en modo normal [15].

```
if (abs(wZAnterior - wZObj1) > 1000) {
      wZObj1 = wZAnterior;
}
if (abs(wZ2Anterior - wZObj2) > 1000) {
      wZObj2 = wZ2Anterior;
}
if (wZObj1 > 800 && wZObj2 > 800) {
      sendData(wXObj1, wYObj1, wZObj1, wXObj2, wYObj2, wZObj2);
      wZAnterior = wZObj1;
      wZ2Anterior = wZObj2;
}
```

Código 8 Filtrado de datos posiblemente erróneos

TCP (transmission control protocol) es un protocolo por el cual el cliente establece una conexión previa con el destinatario antes del envío de paquetes (negociación en tres pasos) y otros métodos con los que asegura que los datos sean recibidos de forma correcta, por lo tanto es un protocolo que **da prioridad a la integridad de los datos** ante la velocidad de transmisión.

UDP (user datagram protocol) es un protocolo que no establece ninguna conexión previa con el cliente ni posee sistemas de control de errores, por lo tanto **no garantiza la integridad de los datos enviados**, sin embargo es muy apto para aplicaciones muy dependientes del tiempo ya que debido a su sencillez no sufre tantos retrasos como por ejemplo TCP. Ya que en nuestra aplicación queremos **transmitir los datos a tiempo real**, con los menores retrasos posibles **se optará por el uso de UDP**, además estaremos comunicándonos entre aplicaciones desde el mismo ordenador, por lo que los problemas de perdida de datos deberían ser mínimos.

Para la implementación del método de envío de datos se ha usado el código de [104] modificando ciertos aspectos para asemejarlo al código de Hasan Azizul [75], por ejemplo se utiliza el **puerto 5005**, en el cual puede ser usado el protocolo UDP [105]. También se utiliza la dirección IP **127.0.0.1** que hace referencia al **localhost** (el propio ordenador), usado para hacer una conexión en el mismo equipo [106].

En caso de tener la opción de ventanas activadas se mostrará en una ventana el video a color y en otra la imagen binaria con los filtros aplicados.

```
cvtColor(colorMat, colorMat, CV_BGR2RGB);
imshow(windowName1, colorMat);
imshow(windowName2, binaryImageMerged);
waitKey(1);
```

Código 9 Mostrar información de video a color e imagen binaria en ventanas

Una vez enviados los datos volveremos a repetir el bucle para obtener la siguiente posición del marcador, en caso de que queramos cerrar la aplicación se saldrá del bucle y **liberaremos los recursos**.

```
depthSensor.destroy();
colorSensor.destroy();
device.close();
OpenNI::shutdown();
```

Código 10 Liberación de recursos antes de finalizar el programa

Para hacer uso de los comandos introducidos por el jugador se hace uso de la función **_kbhit()** y **_getch()**. **_kbhit()** permite que se acceda a **_getch()** solo cuando el jugador haya presionado una tecla, en caso de no incluirlo **_getch()** provocaría que el programa se quedara bloqueado esperando la tecla introducida por el usuario.

```
if (_kbhit()) {
	char k;
	k = _getch();
	cout << k << endl;
	switch (k) {
	case 'e': return 1;  break;
	case 'c': show_video = !show_video; break;
	}
}
```

Código 11 Recepción y tratamiento de datos introducidos en la consola

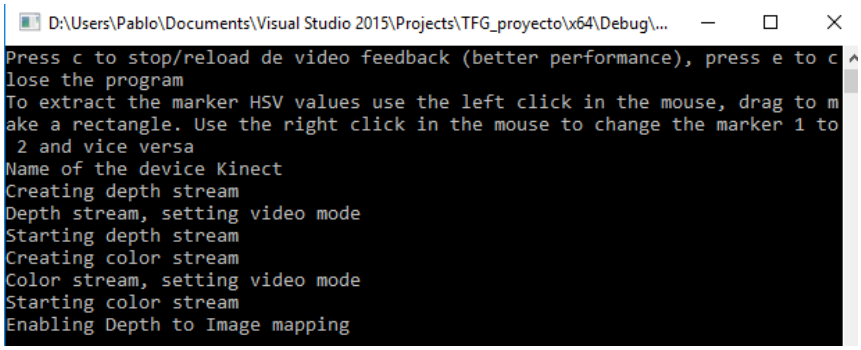


Ilustración 51 Información expuesta en la ventana de la consola al inicio del programa

Explicación juego

Para la estructurar la explicación de una forma más organizada se utilizará el sistema de clasificación usado en los requisitos funcionales del juego.

Recepción y procesamiento de datos

El juego **recibe los datos del programa de seguimiento** utilizando la comunicación por puertos **UDP.** Para el desarrollo de la recepción por parte de Unity de los mensajes UDP se ha utilizado

el código del proyecto de Hasan Azizul [mirar referencia] con ciertas modificaciones para adaptarlo al proyecto (e.g extracción de datos de coordenadas del paquete UDP).

Para lograr la comunicación se hace uso del mismo **puerto** que el programa de seguimiento (**5005**) y la dirección **127.0.0.1** que hace referencia al propio equipo (localhost).

Para permitir la recepción de datos al mismo tiempo que la ejecución del código se inicia un hilo destinado a la recepción de datos.

```
print("UPDSend.init()");
port = 5005;
print("Sending to 127.0.0.1 : " + port);
receiveThread = new Thread(new ThreadStart(ReceiveData));
receiveThread.IsBackground = true;
receiveThread.Start();
```

Código 12 Definición de puerto e inicialización de hilo de recepción de coordenadas

```
private void ReceiveData()
{
      client = new UdpClient(port);
```

Código 13 Inicialización de puerto

Al cerrar el juego se mostrará en la consola de Unity un contador con los mensajes recibidos por segundo, esto permite hacer **pruebas de rendimiento** en la comunicación y procesado de información por parte del programa de seguimiento. Además se eliminará el hilo y se cerrara el cliente, también mostrará si el hilo sigue vivo, lo cual indicaría un error. (Nota: los mensajes Debug.Log se muestran en la consola de Unity no en el ejecutable, sirve para la realización de pruebas en el desarrollo).

```
void OnApplicationQuit()
{
      Debug.Log("Performance: "+iterData/dataTransferTimer+ " Messages
received per second "+"time played: "+ dataTransferTimer);
      if (receiveThread != null)
      {
            receiveThread.Abort();
            client.Close();
            Debug.Log(receiveThread.IsAlive); //must be false
      }
}
```

Código 14 Liberación de recursos e informe de rendimiento al finalizar programa

El paquete enviado por el programa de seguimiento separará las coordenadas mediante el símbolo / por lo que se separará el mensaje por los separadores guardando el resultado en variables que puedan ser accedidas para su posterior procesado.

```
string[] stringSeparators = new string[] { "/" };
string[] result;
result = source.Split(stringSeparators, StringSplitOptions.None);
xPos = float.Parse(result[0]);
yPos = float.Parse(result[1]);
```

Código 15 Extracción y asignación de datos recibidos

El procesado consistirá en **ajustar los mínimos y máximos y normalizar las coordenadas** enviadas por el programa de seguimiento al espacio de juego. Para ello es necesario **definir unos máximos y mínimos** tanto del programa de seguimiento como del juego. Los valores máximos y mínimos de Kinect serán:

Nombre	Valor máximo/mínimo (milímetros)
minKinectX	-800mm
maxKinectX	800mm
minKinectY	-600mm
maxKinectY	500mm
minKinectZ	1000mm
maxKinectZ	1600mm

Los valores se seleccionaron teniendo en cuenta la distancia de las extremidades del cuerpo como por ejemplo los brazos y el entorno en el cual se desarrollaba. Para un mayor ajuste se deberían modificar en caso de usarse en otro entorno.

Aunque se explicará más adelante **el movimiento del sable estará dividido en posicionamiento y rotación**, el posicionamiento utilizará los **mínimos y máximos estáticos** mientras que la rotación irá **modificando sus máximos y mínimos** en la ejecución del juego.

El ajuste de máximos y mínimos consistirá en aproximar los valores que se salgan de rango al mínimo o máximo más cercano.

La normalización servirá para adaptar los datos al espacio de juego. La fórmula que se usará para la normalización será.

Valor normalizado = a + (x - A) * (b - a) / (B - A)

Dónde:

Valor normalizado=minUnity + (valor a normalizar - minKinect) * (maxUnity - minUnity) / (maxKinect - minKinect)

El movimiento del controlador de gomaespuma estará dividido en **dos partes**.

1. Posicionamiento del controlador: se obtiene la **media de las coordenadas de ambos marcadores** y se **posiciona el sable láser en esa situación**. Todos los valores fuera del rango se aproximarán al máximo o mínimo más cercano. Limitar el rango permite que el jugador note más los movimientos de acercar o alejar el controlador. Lo ideal sería obtener el centro del cuerpo del jugador e ir modificando los máximos y mínimos en función del centro del cuerpo, sin embargo para ello se requería el uso del middleware Nite así que se ha optado por mantener los máximos y mínimos estáticos.

2. Rotación del controlador: al contrario que con el posicionamiento en este caso sí que contamos con una manera **de re calcular los máximos y mínimos**. Teniendo en cuenta que el controlador tiene un tamaño de 750mm el valor **máximo** será igual a la **posición media de los dos marcadores + tamaño del controlador/2**, para el cálculo del **mínimo** habría que **cambiar el signo** únicamente. Esto permite que la rotación sea mucho más sensible que en las anteriores versiones, las cuales contaban con un sistema de máximos y mínimos estáticos (e.g si el máximo de profundidad es 4000mm y un mínimo de 800mm a la hora de normalizar los valores no va a dar como resultado la rotación real del controlador).

La ventaja realizar esta reducción de mínimos y máximos es que el controlador se mueve con mucha más **precisión y fluidez**. La desventaja es que los errores se notan más (e.g si no reconoce bien el color de un marcador pueden producirse saltos en la posición o rotación del sable láser).

Una vez se tienen los datos procesados es sencillo hacer el cálculo de **donde tiene que estar posicionado el sable láser y que rotación debe poseer**. En el caso del posicionamiento se deberá también tener en cuenta la posición del jugador para qué el sable se mueva acompañando a este.

```
void SetPos(Vector3 start, Vector3 end)
{
      Vector3 dir = end - start;
      Vector3 mid = (dir) / 2.0f + start;
      sword.transform.position = mid;
      sword.transform.position = sword.transform.position +
player.position - playerIniPos;
}
```

Código 16 Posicionamiento de sable láser

```
…
Vector3 dir2 = endRotation2 - startRotation2;
sword.transform.rotation = Quaternion.FromToRotation(Vector3.up, dir2);
```

Código 17 Rotación de sable láser tras cálculo de máximos y mínimos

Jugabilidad

Jugador

El jugador no tendrá un cuerpo visible, estará compuesto por un **campo que detecte las colisiones de objetos** y estará sujeto a las leyes de la física impuestas por Unity (RigidBody) siendo por ejemplo **afectado por la gravedad**. Contará también con un **campo que evite que el jugador atraviese el suelo**. Aunque no esté asociado al jugador directamente **la cámara** que representará los ojos **estará ubicada en la posición de la cabeza**.

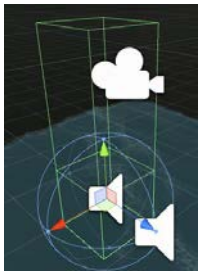


Ilustración 52 Representación del jugador en Unity

Unity 5 implementa compatibilidad directa con varios cascos de realidad virtual incluida la versión DK1, por lo que conectando Oculus al ordenador Unity buscará la cámara principal del proyecto y la moverá en función de los datos ofrecidos por los sensores inerciales. En el caso de las gafas Oculus Rift DK1 solo tomará la **rotación del casco**, en otras versiones de Oculus también aplicará el posicionamiento.

En este caso para mover la cámara para que acompañe al jugador, es necesario hacerla hija de otro objeto ya que no se puede mover la cámara directamente. Mediante la modificación de valores del padre podremos modificar **la posición del padre y de la cámara**.

El movimiento del jugador vendrá dado por los valores del joystick izquierdo o de las teclas w a s d del teclado o las flechas de dirección del mismo.

El jugador rotará en **función de la rotación del eje y** del dispositivo de realidad virtual.

```
VRHMDRotation = InputTracking.GetLocalRotation(VRNode.CenterEye);
transform.Translate(0, 0, Input.GetAxis(verticalAxisName) *
Time.deltaTime * speed);
transform.Translate(Input.GetAxis(horizontalAxisName) * Time.deltaTime *
speed, 0, 0);
float yRotation = headRotation.eulerAngles.y;
transform.eulerAngles = new Vector3(0, yRotation, 0);
```

Código 18 Recepción de datos de entrada de movimiento y aplicación de estos.

Mediante el uso del gatillo izquierdo del mando Xbox o de la barra espaciadora se aplicará al cuerpo del jugador una **fuerza vertical ascendente** en caso de que este haya tocado anteriormente el suelo. Para el la realización del **doble salto** se realizará un proceso similar aunque esta vez se evitará que el jugador salte en caso de que hayan transcurrido más de 2 segundos desde el primer salto o que hayan transcurrido menos de medio segundo (esto último se hace para evitar que el jugador gaste el doble salto al dejar presionado el gatillo).

```
if ((Input.GetButton(jumpButtonName) || Input.GetAxis("LeftTrigger")>0)
&& onGround)
{
      jumping = true;
      int n = Random.Range(0, jump.Length);
      playerAudioSource.clip = jump[n];
      playerAudioSource.Play();
      rb.velocity = new Vector3(0f, jumpValue, 0f);
      timerDoubleJump = 0;
      doubleJump = true;
      onGround = false;
}
```

Código 19 Función de salto del personaje

Al inicio de la partida se inicializará un contador de vida del jugador de **100 puntos de vida** que se verán representado a modo de **barra de vida** en la parte superior del coliseo.

Se permitirá que el jugador **recentre la cámara** a su posición original utilizando la función de realidad virtual **recenter()**, al modificar la rotación de esta **también se verá modificada la rotación del personaje**. Podrá ser activada mediante la tecla “c” o presionando el botón del joystick izquierdo del mando.

```
if (Input.GetKeyDown("c") || Input.GetButtonDown("LeftStickClick"))
{
      InputTracking.Recenter();
}
```

Código 20 Recentrar personaje

Presionando la tecla “h” o el botón select del mando el objeto que representa el robot en caso de estar activado se desactivará o en caso de estar desactivado se activará. La activación permitirá que el robot sea visible y se carguen todas las funcionalidades asociadas a este.

```
if (Input.GetKeyDown("h")||Input.GetButtonDown("SelectButton"))
{
      shooterRobot.SetActive(!shooterRobot.activeSelf);
}
```

Código 21 Desactivar o activar robot

Robot enemigo

El robot **disparará proyectiles al jugador**, en concreto **a la cámara** que tiene el jugador representando sus ojos. Para ello **instanciará un proyectil** el cual es un objeto físico (RigidBody) y le añadirá una fuerza en la dirección de la cámara del jugador. Este proyectil **en caso de impactar** en el jugador **descontara 10 puntos de vida** a su marcador.

Para evitar la sobresaturación del programa por la existencia al mismo tiempo de multiples proyectiles al instanciar un proyectil se inciará un contador que **tras 5 segundos destruira el proyectil**.

El robot cada 2 segundos decidirá si disparar o no, provocando que sea dificil para el jugador adivinar un patrón de disparo haciendole recurrir a sus reflejos (hay un 75% de probabilidad de que dispare).

```
if (shotTimer > shotTime && canShoot)
{
   target = objectToShoot.position;
      if (Random.Range(-1.0f, 1.0f) > -0.5)
      {
            GameObject soundEfectAuxPrefabI =
(GameObject)Instantiate(soundEfectAuxPrefab, transform.position,
transform.rotation);

soundEfectAuxPrefabI.GetComponent<SoundEffectAuxScript>().playAudioSource
("láserShootRobot");
            shotPos.transform.LookAt(target);
            Rigidbody shot = Instantiate(projectile, shotPos.position,
shotPos.rotation) as Rigidbody;
            shot.AddForce(shotPos.forward * shotForce);
      }
      shotTimer = 0;
}
```

Código 22 Función de disparo aleatoria de robot

El robot contará con una serie de valores máximos y minimos que le permitirán **moverse dentro de un rango**. **Cada 2 segundos decidirá si cambiar la dirección** de movimiento en el eje x, y y z. En caso de salir del límite establecido cambiará de dirección a la dirección contraria ignorando el cambio de dirección aleatorio para volver al rango lo antes posible. **En caso de no estar en rango no podrá disparar al jugador** (variable canShoot en el código de arriba).

Sable láser

Mediante la tecla “l” o el botón izquierdo “lb” del mando se podrá **modificar los estados del estado láser** cambiando la activación de los objetos asociados a las hojas del sable láser.

```
if ((Input.GetKeyDown("l") || Input.GetButtonDown("LeftBumper")))
{
      mode++;
      if (mode == 3) mode = 0;
      switch (mode)
      {
            case 0: playerAudioSource.clip = bladeOff;
blade1.SetActive(false); blade2.SetActive(false); break;
            case 1: playerAudioSource.clip = bladeOn;
blade1.SetActive(true); break;
            case 2: playerAudioSource.clip = bladeOn;
blade2.SetActive(true); break;
            default: Debug.Log("ErrorInSelectMode"); break;
      }
      playerAudioSource.Play();
 }
```

Código 23 Modificación de estados del sable láser

Para poder deflactar los proyectiles con la hoja del sable se ha añadido **un campo de colisiones** a esta. Al proyectil se le ha asignado un **material físico de poca fricción** para que rebota al colisionar con otros objetos. El proyectil será destruido en caso de chocar con otro objeto diferente a una hoja del sable láser por lo que el efecto rebote solo afectara a la hoja.

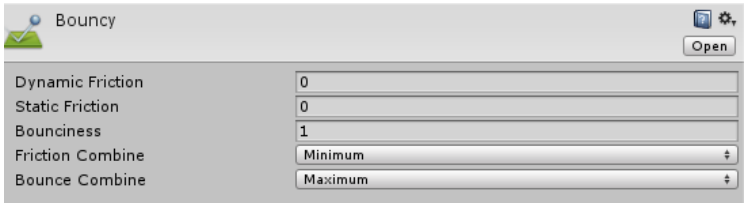


Ilustración 53 Características del material físico de poca fricción

La empuñadura del sable láser contara con otro campo de colisiones pero como antes se ha mencionado en caso de impactar con el proyectil**, el proyectil será destruido** convirtiéndolo en un método alternativo para parar proyectiles.

Al impactar un proyectil en el campo de colisiones de una de las hojas **se incrementará en contador de proyectiles deflactados** modificando el contador expuesto en lo alto del coliseo.

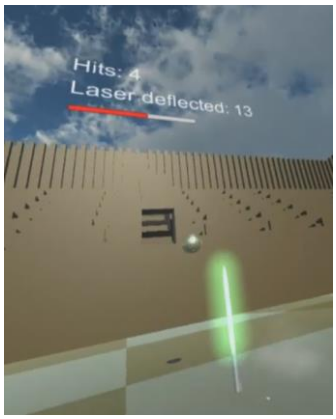


Ilustración 54 Estadísticas de juego en la parte superior del coliseo

Entorno

Para la creación del agua se ha hecho uso de un recurso importado de Unity. Se ha añadido a la altura de la superficie del agua un plano en el que al colisionar el jugador en él provocará su **muerte instantánea** bajando el contador de vida a 0.

El suelo está formado por unas placas que bajan y suben independientemente. Al ir a bajar o subir se informa al jugador con 2 tipos de colores y 3 sonidos diferentes (amarillo va a caer, rojo esta en caída), el número de placas seleccionadas para la caida estará determinado por un temporizador. El número de placas afectadas será **siendo incrementado a medida que vaya pasando el tiempo de juego** añadiendo una dificultad progresiva.

```
void TileFalling(){
      gameObject.GetComponent<Renderer>().material.color=new
Color(1.0F,0.0F,0.0F,2.0F);
      transform.Translate(0.0F, moveUpSpeed * Time.deltaTime *direction,
0.0F);
      if(transform.localPosition.y<=minDistance){
            direction*= -1;
      }
      if (transform.localPosition.y>=maxDistance){
            transform.position = initialPos;
            direction*= -1;
      gameObject.GetComponent<Renderer>().material.color=originalColor;
            timer=timeToFall;
            tileAcopledSound.Play();
              falling =false;
            b_y=true;
            b_r=true;
      }
}
```

Código 24 Función de caída de placa

El comportamiento de subida y bajada estará asociado a cada placa y **será activado por un controlador externo** que guarde las referencias de todas las placas instanciadas del mapa, este también deberá ser el encargado de **crear al inicio del juego el suelo** con el patrón de un tablero de ajedrez.

```
void GenerateTerrain(){
      for (int x = -mapDimension; x < mapDimension; x++) {
            if(tile1){
                  tile1=false;
            }else{
                  tile1=true;
            }
            for(int z = -mapDimension; z<mapDimension; z++){
                  pos = new
Vector3(x*(int)prefabTile_1.transform.localScale.x,-
1,z*(int)prefabTile_1.transform.localScale.z);
                  if (tile1){

      tiles[x+mapDimension,z+mapDimension]=Instantiate(prefabTile_1, pos,
Quaternion.identity)as Transform;
                        tile1=false;
                  }else{

      tiles[x+mapDimension,z+mapDimension]=Instantiate(prefabTile_2, pos,
Quaternion.identity) as Transform;
                        tile1=true;
                  }
            }
      }
}
```

Código 25 Función de creación de suelo con forma de tablero de ajedrez

En caso de que el **contador de vida del jugador llegue a 0** se deberá **transportar al jugador** a una plataforma se aspecto cristalino ubicada en el cielo en la que podrá observar las estadísticas de juego y el aviso de su muerte y **reinicio del juego tras 8 segundos**. Para que el jugador lo primero que vea al morir sean el texto informativo **se aplicará el método recenter()** al morir por lo que además de cambiar la posición del jugador se modificará su rotación dándole el valor de **la rotación original del jugador**, la cual estará orientada el texto informativo.

En caso de que uno de los proyectiles colisione con el robot se concederá la victoria al jugador transportándolo a la superficie cristalina y reseteando la rotación original para que este vea el texto de las estadísticas del juego y el texto informativo de la victoria, **tras 10 segundos el juego será reiniciado**. El juego instanciará varios sistemas de partículas importados de Unity con aspecto de fuego artificial.

```
void setWinningText()
{
      winningText.text = "You won! The time played is: " +
string.Format("{0:0.00}", Time.timeSinceLevelLoad) + ", the game will
restart in 10 sec";
      InputTracking.Recenter();
      player.transform.position = new Vector3(0, 351, 0);
      Instantiate(fireWorks, new Vector3(0, 351, 7), Quaternion.Euler(-
90, 0,0));
…
```

Código 26 Función de victoria

Como antes se ha mencionado en caso de que una placa esté cerca de caer se deberá avisar al jugador **cambiando el color de la placa a amarillo** y reproduciendo un sonido, en caso de que la caída sea inminente **el color cambiara a rojo** además de reproducir otro sonido. Para restaurar el color cuando la placa vuelva a su posición original se guardará color al iniciar la función.

```
if (falling==true){
      timer-=Time.deltaTime;
      if (timer>timeToFall/2&&timer<timeToFall/1){
            gameObject.GetComponent<Renderer>().material.color=new
Color(1F, 0.6F, 0.016F, 1F);
            if(b_y==true){
                  tileYellowSound.Play();
                  b_y =false;
            }
      }
      if (timer<timeToFall/4&&timer>timeToFall/8){
            gameObject.GetComponent<Renderer>().material.color=new
Color(1.0F,0.0F,0.0F,1.0F);
      }
…
```

Código 27 Función de reproducción de audio y cambio de color de placa a caer

Al impactar en el suelo un proyectil se instanciará una **textura de rotura** y unas **partículas de chispas y humo** en la posición del impacto. Ya que el **proyectil será destruido** en el proceso se instanciará un **objeto auxiliar** en el que estarán almacenados varios array de audios y que dependiendo de los valores de la instanciación **reproducirá un sonido aleatorio** de un array o de otro. El objeto auxiliar **será destruido a la finalización del audio**. En caso de impactar en cualquier otra superficie el proyectil no instanciará la textura de rotura.

Ilustración 55 Ejemplo de golpe de proyectil en el suelo

```
public void playAudioSource(string sound)
{
	Start();
	AudioSource audio = null;
	bool dontPlay = false;
	switch (sound)
	{
		case "playerHit": audio = gruntHitArray[(int)Random.Range(0,
gruntHitArray.Length)]; break;
		case "normalHit": audio =
láserShootHitArray[(int)Random.Range(0, láserShootHitArray.Length)];
break;
		case "lightSaberHit": audio =
láserLightSaberHitArray[(int)Random.Range(0,
láserLightSaberHitArray.Length)]; break;
		case "láserShootRobot": audio =
láserShootRobotArray[(int)Random.Range(0, láserShootRobotArray.Length)];
break;
		default: dontPlay = true; ; break;
	}
	if (!dontPlay)
	{
		audio.Play();
	}
	Destroy(gameObject,audio.clip.length);
}
```

Código 28 Función de reproducción de audio en objeto auxiliar

Para que la hoja del sable láser emita luz verde se han asociado a la hoja del sable **cinco fuentes de luz de color verde**.

El **sonido de entorno** estará compuesto por el sonido del **agua** y el sonido del **viento** los cuales se empezará a reproducir al principio de la partida repitiéndose en bucle. El sable láser también cuenta con un sonido de zumbido en bucle que sonará **siempre que este activo**.

Para mejorar la inmersión del jugador respecto a la espada láser se calcula la **velocidad de rotación** a partir de su **rotación anterior** y la **actual**, en caso de superar un límite de velocidad **se elegirá aleatoriamente un sonido** de diez el cual empezará a reproducirse en caso de que no haya ninguno en proceso de reproducción.

```
void applyMovementSound(){
AudioSource audio = null;
Vector3 actualRotation=transform.rotation.eulerAngles;
float speed = ((actualRotation - lastRotation).magnitude) /
Time.deltaTime;
lastRotation = actualRotation;
if (!playingMovementSound&& speed > speedLimit)
{
…
```

Código 29 Aplicación de sonido a sable láser por rotación

7.Conclusiones y posibles ampliaciones

-Video del **prototipo** de abril: https://www.youtube.com/watch?v=n3W4I0sgCVA

-Video de la **versión final** de junio: https://www.youtube.com/watch?v=pCz_zCBTLQw

Tras el desarrollo final del proyecto creo que la solución ha sido bastante satisfactoria, sin embargo para alcanzar este resultado he tenido que abordar **varios tipos de problemas** como por ejemplo:

- **Tratar con un tema tan amplio** como es el de la visión artificial, además de ser una materia de la cual no tenía conocimientos previos.

- **Tener que investigar y comprender el funcionamiento de muchos dispositivos** diferentes para poder extraer funcionalidades que pudieran adaptarse al proyecto.

- **Tanto el uso de librerías como el entorno de desarrollo eran desconocidos para mí**. De hecho gran parte del desarrollo fue retrasado debido a errores del programa de seguimiento.

- Al principio del proyecto **no tenía conocimientos necesarios para decidir qué camino tomar**, por ejemplo creía que Kinect v1 usaba la tecnología Time Of Flight o que la barra sensora de la consola Wii transmitía datos al mando o a la propia consola, también creía que podría realizar el seguimiento de los marcadores únicamente con las funcionalidades del SDK 1.8 de Kinect o más tarde con la librería OpenNI.

- Uno de los mayores problemas de este proyecto y al cual nunca me había enfrentado en la carrera era que **el problema que trataba de solucionar podía no tener solución**. Por ejemplo, podía darse el caso de que no fuera capaz de conseguir que el programa de seguimiento procesara las imágenes a la velocidad necesaria o que no hubiera manera de comunicar Kinect con el juego.

- **No se tenían todos los recursos necesarios para generar el juego** como por ejemplo los modelos 3D o el audio, de modo que también he tenido que aprender a usar varios tipos de programas como por ejemplo Blender.

- El proyecto **abarcaba varios temas que iban siendo actualizados a lo largo del desarrollo del proyecto**, por ejemplo durante el desarrollo del proyecto fueron saliendo al mercado varios dispositivos de realidad virtual o fueron saliendo nuevas noticias de sus especificaciones.

- Era un problema amplio por lo que me atasqué al principio tratando de reunir toda la información necesaria para definir todos los requisitos necesarios (más tarde descubrí que para este tipo de proyecto lo mejor era utilizar técnicas ágiles).

¿Qué aporta este proyecto?

- **Un amplio estudio del estado del arte de los dispositivos más relevantes del momento** tanto de controladores de movimiento como de dispositivos realidad

virtual. Para lograr una solución interesante creía que no era suficiente con explorar únicamente los dispositivos que se iban a utilizar en el proyecto si no que había que contemplar el problema desde el mayor número de puntos de vista posible extrayendo las funcionalidades más interesantes y adecuadas para el proyecto. He intentado además que la explicación de la historia y el funcionamiento de los distintos dispositivos sea lo más interesante posible intentando entrelazar los productos mediante las funcionalidades compartidas o similares o mediante la historia del desarrollo del producto y de cómo los dispositivos han ido actualizándose dependiendo de los avances en otros productos.

- **Un estudio de los distintos tipos de problemas que conlleva el uso de realidad virtual**, su explicación y diferentes formas de solventarlo. Por ejemplo, el mareo producido en el usuario por el uso de la versión DK2 de Oculus es inferior al de la versión DK1 debido al incremento de la tasa de refresco y al uso de la tecnología "low persistence". Se ha dedicado tiempo en comprender los diversos problemas en vez de simplemente mencionarlos para poder ofrecer una explicación lo más clara posible además de mostrar posibles soluciones. Por ejemplo en caso de que el jugador se mueva por el entorno lo mejor es no prefijar su movimiento, como por ejemplo una simulación de una atracción, si no permitir que sea el jugador el que esté a cargo de todos sus movimientos. Otra de las soluciones al mareo explicadas en el desarrollo del proyecto ha sido la utilización de puntos fijos permitiendo al usuario orientarse en el entorno virtual.

- **Se demuestra que es posible hacer uso de Kinect V1 para Windows para el seguimiento de objetos externos al cuerpo humano** añadiendo a la cámara una funcionalidad extendida.

- **Se demuestra que el sistema elegido opera de forma fluida y rápida**. Por ejemplo para este proyecto se ha utilizado OpenCV debido a gran optimización de código y UDP como protocolo de comunicación entre el programa del seguimiento y el juego. Se han obtenido unos resultados en el mejor de los casos de una media de 30 actualizaciones por segundo de la posición del controlador alcanzado la frecuencia máxima dadas las características de Kinect que puede obtener un máximo de 30 imágenes por segundo. La tasa de actualización del juego también ha demostrado ser óptima logrando alcanzar una tasa de refresco superior a 60 FPS lo que en términos de características de Oculus Rift DK1 es el mejor resultado posible ya que el dispositivo de realidad virtual posee una tasa de refresco de 60 Hz de forma que más de 60 FPS no se apreciarían.

- **Se demuestra que es posible hacer uso de versiones de dispositivos antiguas para agregar nuevas funcionalidades**. Por ejemplo la versión utilizada Kinect v1 for Windows ha sido sustituida por la versión 2 la cual ofrece mucha mejor calidad, sin embargo he intentado demostrar mediante este proyecto que pueden seguir siendo usadas para una amplia variedad de proyectos como por ejemplo proyectos basados en el seguimiento de controladores como este mismo. Creo que el ciclo de vida de estas cámaras aún no ha terminado, por ejemplo también podrían servir como apoyo de otros sistemas más actuales como por ejemplo usarse junto con el dispositivo HTC Vive para hacer un seguimiento del esqueleto del jugador. Aunque no he podido

probar si el programa funciona con la cámara Kinect para Xbox 360 creo que podría ser compatible con esta aplicación ya que dispone de unas características muy similares a Kinect v1 para Windows. La ventaja que conllevaría hacer uso de ese tipo de cámaras sería que son mucho más abundantes que sus versiones de PC, recordar que Kinect para 360 entró en el libro Guiness de los Records por vender 133.333 unidades al día durante 60 días lo cual deja una amplia cantidad de cámaras sin usar que podrían ser utilizadas para mejorar proyectos más actuales (aun así hay que tener en cuenta que la utilización de cámaras Kinect 360 está restringida, véase marco legal). Otro producto con varios años de antigüedad es la versión Oculus Rift DK1 sin embargo mediante este proyecto se demuestra que es posible hacer uso del dispositivo para el desarrollo de juegos actuales. Aunque ofrece por ejemplo la característica de posicionamiento de las versiones superiores creo que se podría solucionar este problema con el uso de marcadores de una forma similar a la planteada por Sony en su PlayStation VR (o con el propio programa de seguimiento creado).

- **Se demuestra que es posible transmitir datos desde una cámara Kinect a motores de videojuegos como Unity de una forma muy sencilla y rápida mediante el uso de un puerto UDP**. Además también usando este método se puede desarrollar en paralelo dos aplicaciones en entornos de desarrollo diferentes.

- **Se plantean numeras mejoras y alternativas**. Esto ha sido posible gracias al amplio estudio del estado del arte realizado.

- **Este trabajo permite orientar a otras personas que quieran realizar desarrollos similares**, lo cual es bastante interesante debido a que este proyecto abarca muchos tipos de alternativas. He intentado dejar claro que recursos me han sido de mayor utilidad dentro del proyecto y el porqué de las elecciones tomadas de forma que si otra persona está interesada podrá comenzar su proyecto de una forma más fácil y con menos incertidumbre que yo, pudiendo dedicar menos tiempo a la búsqueda de información básica pudiendo dedicar más tiempo a la creación de proyectos de mayor complejidad.

- **La lectura de este trabajo facilita el uso e instalación de las librerías OpenCV y OpenNI en Visual Studio 2015**. Ya que se perdió mucho tiempo tratando de instalar y configurar el entorno de desarrollo se ha añadido un anexo con una guía para la instalación de las librerías con un pequeño test al de cada explicación para verificar su correcta instalación.

Metas y logros a nivel personal

- **Ser capaz de adaptarme a las características de un proyecto tan amplio y con tantas incógnitas**. El desarrollo de este proyecto ha sido muy diferente de los realizados hasta ahora pero creo que he sido capaz de elegir el mejor método con el que lidiar con el desarrollo del proyecto como puede ser la elección del uso de técnicas ágiles frente a métodos más comunes como el ciclo de vida en cascada. La elección de un método iterativo e incremental me ha permitido poder ir realizando el proyecto en pequeñas porciones permitiendo añadir poco a poco varios tipos de requisitos que me hubieran sido imposibles de concebir al principio del desarrollo.

- **Ser capaz descomponer un problema complejo en partes más asequibles**. Se ha conseguido paralelizar el desarrollo y pruebas mediante la división del proyecto en dos partes independientes (programa de seguimiento y juegos). También creo haber administrado de una forma correcta los tiempos disponibles en función de la dificultad asociada. Por ejemplo, se asignó al desarrollo del programa de seguimiento la mayor franja de tiempo pero conseguí también reservar suficiente tiempo para terminar el juego.

- **Se ha seguido de forma bastante aproximada la planificación inicial,** lo cual creo que demuestra una buena estimación de tiempos, especialmente teniendo en cuenta problemas asociados al desarrollo del proyecto. Se ha conseguido entregar los dos informes de seguimiento en fechas cercanas a las propuestas y creo que han sido claros y de utilidad. En el primer hito me centré en demostrar que la realización del proyecto era posible y qué posibles mejoras podía realizar en este para alcanzar un resultado satisfactorio, para ello se adjuntó una sintetización de las partes clave del funcionamiento del proyecto y un video que mostrara las funcionalidades de este. En el segundo hito se realizó otra sintetización de las principales funcionalidades y se hizo hincapié en el porqué de los cambios y de las mejoras realizadas, asegurándose de haber cubierto todos los puntos de las propuestas surgidas a raíz del informe de seguimiento (I).

- El haber ido buscando la mejor aproximación para abordar el proyecto y el haber ido solucionando los problemas a medida que se iban generando creo que me ha demostrado una **gran capacidad de autonomía** lo cual creo que es una característica clave para un Ingeniero Informático.

Objetivos no alcanzados:

- La implementación de filtros para la estabilización del movimiento del sable láser. Aunque lo intenté no conseguí implementar una solución satisfactoria. El problema es que si el reconocimiento de color del programa de seguimiento falla es posible que el movimiento del sable láser se vuelva inestable. Se comentará más en profundidad a continuación como una de las posibles ampliaciones del programa de seguimiento.

- Implementación de compatibilidad con ficheros de video .ONI. Como ya se ha explicado en el desarrollo de la solución los ficheros de .ONI almacenan los datos de la cámara a color y del mapa de profundidad de forma que permiten sustituir a la cámara para la realización de pruebas. Aunque se intentó implementar esta funcionalidad se descubrió que la cámara Kinect no permitía alinear los datos de la cámara a color con los datos del mapa de profundidad ya que era un video ya creado. Ya que no era una funcionalidad básica se acabó descartando su implementación. Sin embargo se cree que en caso de haberse conseguido se podrían haber hecho las pruebas de una forma más rápida y sencilla.

A lo largo del desarrollo se han podido apreciar numerosas posibles **mejoras y ampliaciones de la solución**. Se cree que haber separado el proyecto en dos partes independientes (programa de seguimiento y juego) facilitaría de forma notable la implementación de algunas de estas.

Las mejoras u ampliaciones propuestas para el **programa de seguimiento** son:

- **Aumentar la cantidad de filtros** para un mejor reconocimiento del color de los marcadores, por ejemplo, en caso de tener un objeto de color similar al de un marcador, siendo el objeto más grande que éste, puede originar fallos en el seguimiento. La solución propuesta es **implementar filtros de eliminación del fondo**. Esto puede ser realizado de varias maneras, una por ejemplo es de **tratar de dar mayor peso a los objetos que se muevan**, de un frame a otro se extraen los cambios producidos en las imágenes a procesar y aquellas posiciones que suelan tener el mismo valor se les dará menos peso. Otra opción sería aprovechar el mapa de profundidad y **eliminar aquellos datos a cierta profundidad**, o si quisiera hacer de forma dinámica calcular las coordenadas del centro del usuario y eliminar aquellos datos detrás de este. OpenCV también implementa varios tipos de eliminación de fondo. Se cree que añadir este tipo de filtro podría ayudar **a incrementar la precisión del programa de una forma notable**, especialmente al ejecutar el programa en ciertos tipos de entorno.

- Se podría tratar de utilizar **otra forma de seguimiento del objeto**, por ejemplo un seguimiento mediante el uso de marcadores del mismo color. **Los marcadores deberían formar un patrón** para poder **identificar** cada marcador **individualmente**. Esta idea fue pensada tras comprobar cómo funcionaba el sistema Structured Light (SL) de Kinect y algunas de las soluciones propuestas de los prototipos HTC Vive. Un ejemplo de su posible aplicación al proyecto es utilizar tres tiras de goma eva del mismo color en el controlador de gomaespuma, dos serían situadas en la mitad superior separadas por un espacio, y la otra alejada en la mitad inferior, de forma que ya tendríamos un patrón. A la hora de identificar individualmente a cada marcador **se necesitaría la visibilidad de los tres**. El superior sería aquel que tuviera un marcador cercano y estuviera más alejado del no cercano etc… **Esta solución es ampliable a patrones más complejos** que permitirían asegurar la identificación de los marcadores aun en caso de que parte de estos no estuvieran visibles por la cámara.

- No limitarse únicamente al seguimiento de los marcadores sino también **añadir un seguimiento de las distintas partes del cuerpo**. Aunque el objetivo principal de este proyecto era **verificar la posibilidad de usar controladores externos**, también podrían ser usadas las funciones para las que está dedicado el uso de la cámara Kinect que es el seguimiento de juntas del cuerpo. Esto puede ser logrado por ejemplo mediante el uso del middleware NITE2 junto con OpenNI2 y permitiría generar por ejemplo un **cálculo de mínimos y máximos dinámico de la posición del controlador** respecto a la posición del jugador. Otra posibilidad que podría generar su uso sería la de **representar el cuerpo del jugador en el juego**, permitiendo que este pudiera esquivar los proyectiles con el movimiento de su propio cuerpo.

- Añadir una **compatibilidad con cámaras estándar** eliminando la necesidad de usar una cámara Kinect, es decir utilizar OpenCV para intentar extraer la profundidad y la posición relativa de los controladores, esto se podría lograr por ejemplo haciendo uso de un sistema similar al usado por PlayStation Eye, el cual a partir del tamaño de un marcador es capaz de posicionarlo en el espacio 3D calculando el tamaño de este.

- Añadir la **posibilidad de usar dos cámaras Kinect**, posicionando una enfrente del jugador y otra detrás de forma que se pudieran **minimizar** los casos en los que los **marcadores dejaran de estar visibles** por alguna de las dos cámaras. En la explicación del funcionamiento de las cámaras Kinect se explica por qué este sistema es **difícil de implementar**, sin embargo se han encontrado ejemplos [107] en el que se ha aplicado satisfactoriamente este método así que se podría considerar intentarlo.

Las mejoras u ampliaciones propuestas para el **juego** son:

- **Añadir filtros para estabilizar el movimiento**. Aunque este caso se intentó aplicar al proyecto no se consiguió obtener un resultado satisfactorio. El problema es que el programa de seguimiento puede ofrecer unas coordenadas erróneas debido a problemas de iluminación o fallos a la hora de reconocer el marcador produciendo **saltos bruscos en el movimiento del sable láser**. Por ejemplo, si el programa de seguimiento detecta cantidades similares de color verde en las posiciones superior e inferior de un marcador pero no en la mitad, las coordenadas centrales del marcador verde se irán alternando entre la parte superior y la parte inferior provocando varios pequeños saltos en el movimiento del sable láser. Se podría introducir una serie de condiciones que **al detectar ese tipo de comportamiento anómalo lo ignorara** estabilizando el movimiento del sable láser.

- **Realizar otro tipo de juego o añadir nuevas funciones a este**, antes de crear el juego final se pensaron otras posibilidades. Una las ideas, cuya imagen del prototipado puede ser visualizada en el planteamiento de la solución, consistía en la creación de un **nivel lineal** en un complejo industrial o nave espacial haciendo que el jugador fuera aprendiendo poco a poco las distintas mecánicas del juego. En ese caso por ejemplo se planteaba el uso de un botón a modo de fuerza que empujara los objetos situados en frente del jugador pudiendo ser utilizados para dañar al robot. Aunque la idea fue descartada sería interesante probar su aplicación ofreciendo al jugador más niveles que progresivamente fueran incrementando la dificultad del juego.

- **Mejorar aspectos gráficos**. Aunque se está satisfecho con los resultados obtenidos, teniendo en cuenta el tiempo con el que se contaba, se podría intentar mejorar varios aspectos. Por ejemplo se podría intentar **mejorar el shader de iluminación del sable láser** para que ofreciera un aspecto **más realista**, se podría también añadir una funcionalidad que permitiera **modificar el color de la hoja del sable láser**. Otra mejora sería la del entorno, sustituir los **modelos** 3D por otros **más detallados**, añadir más detalles al coliseo y al entorno en general (e.g estandartes, antorchas, estatuas, publico…) que ayudaran a conseguir una **mayor inmersión del jugador**. Se podría texturizar también los modelos 3D, por ejemplo el estadio, diferenciando aún más las diferentes partes de este.

- **Mejorar e incrementar el audio**. Aunque se cree que se ha conseguido implementar una amplia variedad de sonidos que **eviten la sensación de repetición al jugador**, sí que se cree que es posible **añadir más tipos de sonidos** que permitan aumentar la sensación de inmersión del jugador. Por ejemplo se podría añadir varios sonidos de pasos al moverse, el producido por la piedra al caer o saltar sobre ella, el sonido de salpicadura del agua al caer el jugador etc…

Otras mejoras y otros posibles proyectos similares podrían ser:

- **Mejorar el controlador de gomaespuma**, aunque el controlador creado ofrece de unos resultados bastante precisos y su coste es muy reducido, se cree que es posible hacer una mejora aumentando la calidad de los resultados del seguimiento. Por ejemplo, hay casos en el que los marcadores por temas de iluminación no son detectados de forma correcta por la cámara, especialmente la parte superior del marcador (i.e la tapa superior del controlador de gomaespuma). Se cree que **usar una superficie esférica en vez de una cilíndrica como marcador** podría incrementar la calidad del seguimiento. Esto se planteó como idea en el desarrollo del proyecto y se intentó llevar a cabo, para ello se compraron 2 bolas de poliestireno para ser forradas con goma eva y unidas con una varilla de madera. Se acabó descartando ya que la varilla de madera podría provocar daños a las personas adyacentes al jugador o al mobiliario cercano, sin embargo se cree que sería interesante comprobar la eficacia del programa de seguimiento con controladores similares. Otra mejora interesante podría ser **implementar** en el controlador mediante Arduino **varios sensores inerciales** que ayudaran a calcular el movimiento del controlador en caso de que la cámara no tuviera visibilidad de los marcadores.

- **Hacer uso del seguimiento de marcadores para otro tipo de aplicaciones**, por ejemplo para el sector educativo. Se podría intentar **crear una pizarra interactiva** usando un marcador a modo de bolígrafo y pudiéndose medir la presión aplicada utilizando la profundidad de dicho marcador. Otra posibilidad sería la de **realizar dibujos en 3D**, haciendo uso de un mando en una mano para manejar la paleta de colores o el tipo de pincel por ejemplo y un marcador como pincel (una aplicación similar sería Tilt Brush by Google).

- Se podría intentar hacer una **emulación de otros dispositivos de control de movimiento** como los controladores del HTC vive, permitiendo **sustituir los datos ofrecidos de los controladores** por los de un mando inalámbrico y el controlador de gomaespuma. Evidentemente el posicionamiento no sería tan preciso como el sistema utilizado por los controladores HTC Vive, pero permitiría **un acceso más barato a juegos de realidad virtual**. Esto parecía interesante así que se intentó en un momento del desarrollo del proyecto accediendo a código del juego “The Lab by Valve” para comprender como pasar otros valores al juego, como por ejemplo los producidos por el programa de seguimiento. Debido a la complejidad del problema se acabo abandonando la investigación, sin embargo se cree que es posible.

- Intentar **usar como controlador un dispositivo móvil**. Al principio del proyecto **se descartó el uso del móvil** ya que aunque contara con varios sensores inerciales (giroscopio acelerómetro y magnetómetro) el error acumulado por estos **no permitía un movimiento lo suficiente preciso**. Sin embargo este problema **podría ser solucionado con el uso de la cámara del móvil**, ubicando varios marcadores en un punto fijo y **aplicando una técnica de triangulación** similar a la usada por el WiiMote. De esta manera se podría usar la triangulación para **corregir el error** producido por los sensores inerciales **incrementando la precisión del posicionamiento** 3D. El reconocimiento de marcadores puede ser implementado en el propio móvil ya que es posible la implementación de la librería OpenCV en dispositivos Android. La

comunicación de coordenadas al ordenador podría ser realizada mediante Bluetooth por ejemplo.

Summary

Introduction

This is a really exciting year to such fields as video game development, computer vision, virtual reality and motion controller devices.

The video game industry is currently experiencing a boom time, the existence of video game sales platforms such as Steam or GoG allow an easier and cheaper way to distribute games. Another important factor is the presence of crow funding platforms like KickStarter which allows **that small developers can expose their ideas and get funding to develop their games.**

Since the success produced by the Wiii console in 2006, many companies have tried to exploit the niche market through the creation of several motion controllers. The mechanics of some devices works thanks to the **computer vision techniques**. While Sony tried to track the position of their motion controller PlayStation Move with their camera PlayStation Eye, Microsoft was taking some different approach with the camera Kinect by, in Microsoft Words "eliminating the need for controllers and turning you into the controller with your whole body". Both had different approaches but they share the use of **computer vision techniques** as their main mechanic. These techniques have been evolving and adapting over the years. For example Sony mechanics for the positioning of their motion controller have been used as a main method to tracking and positioning their virtual reality device PlayStation VR.

The video game market is also being shaken by the **arrival of several virtual reality devices**, in particular through this year some of the most famous VR devices has come to sale such as the Oculus Rift consumer version (CV1) or the system developed by Valve HTC Vive, which also include two high-precision motion controllers.

Other companies such as Sony have already announced the arrival of its own virtual reality devices in October 2016. Also companies such as Microsoft which doesn't produce virtual reality devices have shown interest in the market announcing a new console for 2017 with the characteristics necessary to withstand virtual reality games.

The **computer vision** field is also more active than ever, **the need to treat large amounts of data** such as images and videos has led many companies to show interest in this field. Using techniques from computer vision and artificial intelligence Microsoft for instance has released a program that tries to recognize people's emotions from an image based on theirs facial expressions. Other application out to the public this year also brought by Microsoft is CaptionBot who tries to describe the images uploaded by the user.

There is also a wide variety of companies interested in the use of computer vision to improve their products such as applications for security, implementations in the medical field or for example improve the product's marketing using consumer analysis. The automotive market has also been very interested in the use of computer vision, it is a fact that more and more companies are trying to lead the creation of an autonomous car, including companies such as Apple, which has announced the creation of an autonomous car for the year 2021 approximately. Companies like Tesla Motors try to achieve this by **using cameras** as their primary method to ensure the autonomy of their cars. The camera approach is used also for instance by the independent developer George Hotz which is trying to develop a low cost equipment to transform standard cars into autonomous cars.

For this project my current tutor Yago Saez asked me to make a **light saber simulator** getting the data in **real time**. He also asked to apply **virtual reality** in the simulation to accomplish a more immersive experience for the user. He also let me choose the best approach to deal with the problem.

After a preliminary study I decided to make use of the Kinect camera V1 for Windows along with the virtual reality headset Oculus Rift DK1 both provided by the computer science department of the University Carlos III of Madrid. The idea was to **create a low cost motion controller** tracked by **computer vision techniques** and be capable of transmitting the controller movements to a game trying also to ensure a smooth and accurate experience for the player. Also I needed to make the game compatible with the virtual reality device to improve the player immersion.

The development of the solution however raises several problems. For example Kinect implements several algorithms for capturing movements of the body, however does not include any kind of implemented functionality that allows you to **recognize and get the position of external objects**. To solve this problem I needed to create a program that uses external libraries for **image processing**.

Other problems that arise are: The tracking system will be able to run **smoothly**? **How often it can update** the position of the lightsaber? Which limitations provide the use of the Kinect camera? There are **other ways** to solve the problem? With which degree of **accuracy** can be represented the movement of the physical controller? Is it possible to obtain the information necessary for the positioning and rotation of the light saber **without the use of inertial sensors**?

In the case that the tracking application works, which is the best way to apply the results in a game to accomplish a **nice player experience**? What game mechanics can be applied by adding the virtual reality headset? What problems are associated with the use of virtual reality and how to deal with them?

I made this bachelor thesis to answer these questions, to accomplish that I will study how some of the most famous motion controllers and virtual reality devices works. Studying these devices and his history I will try to learn which possible techniques could be applied to a satisfactory resolution of the problem.

Summary development

There are several ways to track a controller, so if I wanted to make one I had to study the functionality of the most relevant motion controllers.

Wiimote is the main remote in the Wii console device. Its components include a **3-axis accelerometer** and an **infrared sensor.**

The accelerometer can calculate the tilt of the remote, but it can't get the data from the horizontal axis (yaw axis). To solve this problem Wii have a sensor bar composed of 5 infrared LEDs on each side. With these 5 infrared LEDs the infrared sensor in the remote sensor is able to triangulate the position of the remote and also get his rotation. When the sensor bar is not visible by the remote it use the accelerometer to calculate movement, however the more time without visibility from the sensor bar, the more error will be the accumulated. When the sensor has some of the LEDs in vision the remote will correct his cumulative error.

PlayStation Move is a motion controller created by Sony Computer Entertainment. To obtain the position and rotation of the remote it uses some inertial sensors (accelerometer, gyroscope and magnetometer) and a camera.

The remote control has a sphere at the top containing a RGB LED that allows changing his color. To keep track of the remote control the camera detects the color of the sphere and get the coordinates x, y. To calculate the depth the camera uses the size of the sphere (if the sphere is smaller than before the depth value will be greater).

The remote control uses the inertial sensors to calculate its own rotation and the camera to correct the cumulative error.

Kinect is a device created by Microsoft. Its main function is to detect the movement of the body of the player. There are several versions of Kinect, version 1 makes use of technology Structured light (SL) which can calculate a depth map from the surroundings by using an infrared projector and an infrared camera. Depth is calculated by projecting a pattern with the infrared projector and calculating the deformations suffered by the pattern. Since the pattern is known and does not repeat sequences it is possible to identify each point individually using the adjacent neighbors. The sunlight can erase these patterns. Kinect also uses the RGB camera to track the player's skeleton.

Once the study of the devices is already done I can choose now how I want to solve the project

One of the ideas provided to me by my tutor at the beginning of the project was to use a smartphone as a controller using **the smartphone sensors**. Inertial sensors used by a smartphone are similar to those used in the WiiRemote so I decided that the use of **a smartphone was not suitable for the resolution of the problem**. The reason is because the inertial sensors accumulate error over time, the Wii remote can use its infrared sensor to remove the error accumulated by the inertial sensors but the smartphone no.

It was also discarded the use of the WiiMote since the player would need occasionally direct the remote control towards the sensor to correct the accumulated error. In addition the accelerometer of the WiiMote could not calculate lateral orientation only tilt so the controller would need to have vision of the sensor bar to calculate the horizontal rotation (yaw).

After studying the functionalities of PlayStation Move and Kinect I decided to use computer vision to deal with the problem. My goal was to create a program that could track 2 different colored markers and extract their 3D coordinates. Then, send the coordinates to the game and obtain from them the movement and rotation of the lightsaber.

Camera Kinect can extract a color video to follow the markers and a depth map to calculate its depth so I chose Kinect v1 for Windows as my tracking device.

To access the Kinect data I tried the SDK (software development kit) official of Kinect (v 1.8). However after a study of its different features I discovered that it didn't have any methods that would allow me to keep track of an external object. To solve the problem I decided to use the Visual Studio 2015 development environment adding the OpenNI library to access data from the Kinect camera and the OpenCV computer vision library to process the data.

One of the problems that had to use Kinect was that refresh rate of 30 Hz, allowing obtaining a maximum of 30 images per second. Since I did not know at what speed the program could

process data I decided to try to get an update rate for the position of the lightsaber by 20 to 30 times per second.

The game engine chosen was Unity since it offered a simpler way to implement the game than other game engines such as CryEngine or UDK (Unreal Development Kit).

To communicate the tracking program with Unity I decided to make use of a UDP port since it offered a very high data transfer rate.

I decided that to make the tests and send progress to the tutor I would make videos that demonstrate the operation and viability of the project.

In mid-April i created a prototype of the tracking program and the game that worked well. Then I recorded the video of the application functionality and sent it to the tutor with a short explanation.

Later the prototype of the tracking system was improved to achieve faster and more accurate tracking of the markers. The process of the tracking system is this:

- The Kinect camera is initialized and configured via OpenNI.
- With the extracted data the program tracks colors, in this case I have used a half pool noodle with two pieces of rubber eva at the ends.
- To track colors the user must define the maximum and minimum values HSV (hue, saturation, value) of the desired colors to follow.
- The program search in the image color obtained by the camera RGB values between the maximum and minimum HSV defined.
- Then the program uses the positions of the conflicting values to generate a binary image where the color white (255) represents a value within the range and the color black (0) other values.
- Morphological filters are applied to the binary image to remove noise
- The system try to get the edges of white pixels groupings.
- The program then calculates the area of the interior of the edges and choose the larger area.
- The program then calculates the center of the greater area and search in the depth map the depth in millimeters of the pixel in the center of the area, it also calculates the coordinates x and y in millimeters.
- Once the application has the coordinates of the 2 colors it check if there is not any outlier and proceeds to send the data by a UDP port. The game will get the data, process them and apply them in the game.
- The data from both images, color and a fusion of the binary images of the two markers, will be displayed in different windows through the execution of the application.

- The program will use a window for the introduction of commands from the user allowing him to close the application or avoid updating the information displayed in the windows to achieve higher performance.

The HMD (head mounted display) was used to implement virtual reality in the game. The device was Oculus Rift DK1 which can get its own rotation from the inertial sensors and with a refresh rate of 60 Hz, making 60 FPS the best refresh rate for the game which the player could see.

To develop the game I thought several ideas but in the end I decided to follow this approach:

The game will use a Xbox 360 controller as a secondary input device for such things like moving the player, jumping, etc…

The game takes place in a coliseum. In the central part of the coliseum there is a ground formed by tiles and under these tiles there is water. Every certain period of time a group of tiles will be selected and start to fall through the water then them will rise again. The number of selected tiles will be increased over time, challenging the player over time. When the player touches the water he will lose the game.

The coordinate values obtained by the monitoring program must be processed so I defined some minimum and maximum values for the received data and the game data.

The game will **adjust maximum and minimum** data approaching to their closest minimum or maximum.

To adapt the received data to the game space the game will make a **standardization** of the received values.

The movement of the lightsaber will be divided into **two parts**:

- **Positioning the lightsaber**: Gets the **average value from the coordinates of both markers** and transform the **lightsaber position to the average point**. All values outside the range are rounded to nearest minimum or maximum. We don't have the center of the player so minimum and maximum values must be static, for example if I have a depth maximum of 2000 millimeters and the marker is at 3000 mm it would be rounded to 2000 but if I could calculate the center of my body I could modify dynamically the depth maximum to 3500 mm for example.

- **Rotating the lightsaber**: Unlike positioning **the maximums and minimums are dynamic**. Knowing that the controller has a size of 750 mm, the **maximum value** should be equal to the **two markers average position + size of the controller/2**. The calculation of the **minimum value** would have to **change the sign + to -**. This allows getting an accurate rotation because having the controller size allow the program to adjust the maximum and minimum values with great precision.

The player will not have a visible body, it will consist of a **field that detects collisions of objects** and he will follow the laws of physics imposed by Unity (RigidBody) for instance being **affected by gravity**. It will also have a **field that avoid that the player passes through the tile ground**. **The camera** will be the eyes of the player.

The camera will be rotated by virtual reality device inertial sensors allowing the player to look around rotating his head.

The movement of the player shall be given by the values of the left joystick or the keys w a s d or the arrow keys.

The player will rotate along the virtual reality device **y axis rotation**.

Using the left trigger in the Xbox 360 remote or the SPACEBAR will make the **player jump** (if he previously touched the ground). The player can also perform a **double jump** after the first jump.

The player starts the game with 100 health points. The player health will be shown to the player as a health bar located in the sky.

The player will **recenter the camera** to its original position by pressing the "c" button or pressing the left stick on the Xbox 360 remote.

In the game there is a robot that shoots projectiles to the player. The player must dodge or deflect them since they will discount10 health points from the player's health. When the player's health reaches 0 he will lose the game.

The robot will shot and move randomly so the player must rely on their reflexes.

The projectiles will be destroyed if they collide with any object that is not a lightsaber blade. Where the projectile hit a lightsaber blade the projectile will be deflected. To achieve this I made a physical material in Unity with low friction.

The player can use the lightsaber to stop or deflect the projectiles. When one of the projectiles hit the robot the player will win the game.

By pressing the key "h" or the "select" button on the Xbox 360 remote will deactivate or activate the robot making him disappear or appear.

By pressing "l" or the left button "lb" on the Xbox 360 remote you can **modify the states of the lightsaber** to: one blade activated, two blades activated, zero blades activated.

At the top of the stadium floating in the sky along the health bar there will be a text which will report the game statistics (hits received by the player and deflected projectiles).

Losing or winning will transport the player to a crystal surface in the sky that will allow him to view the game statistics and an informational message of victory or defeat. The game will restart after a few seconds. In case of victory the game will also create some fireworks around the text.

To improve the game experience some actions have a sound to play when they are triggered like the players actions such as jump, fall, receiving damage, switching on or off the lightsaber, rotate the lightsaber, or game actions like the robot firing, tiles falling, collision of the projectile, fireworks or also for environmental sounds as the sound of the wind or the sound of the water.

If a projectile impacts an object it should show particles of sparks and smoke, when it hit the ground it will instantiate a ground rupture texture.

Resources like water or particles like Sparks, smoke and Fireworks have been extracted from Unity, however for this project it has been necessary to create multiple resources.

I've had to create and edit various sound effects such as for example the player jump sound, hurt sound or land sound. I made use the free tool Audacity to record and edit several types of sounds from each class to avoid the feeling of repetition. For instance I recorded five different grunts for the damage sound.

I also had to make 3D models for the game so I used the free tool Blender.

I've done several tests to check the application speed and I got a great results. For example the tracking system was able to achieve an update rate of 26 messages per second received by the game and a rate of 30 in case the user choose the option to doesn't show video and binary images data in the tracking system. **Achieve 30 position updates per second is the maximum allowed by Kinect** since it can extract a maximum of 30 images per second.

I also managed to achieve a refresh rate higher than 60 FPS in the game so that a higher refresh rate would have no impact since Oculus Rift DK1 virtual reality device shows a maximum 60 images per second to the player.

Conclusion and possible extensions

-Video of the **prototype** of April: https://www.youtube.com/watch?v=n3W4I0sgCVA

-Video of the **final version** of June: https://www.youtube.com/watch?v=pCz_zCBTLQw

After the final development of the project, I believe that the solution has been quite satisfactory, however to achieve this result I've had to deal with **various types of problems** such as:

- **Dealing with such a broad subject,** for instance machine vision which I had no previous knowledge.

- **Having to investigate and understand the functionality of many different devices** to extract features that I could take for the project.

- **Both the use of computer vision libraries and the development environment were unknown to me**.

- At the beginning of the project, **I had no knowledge to decide which path to take**, for example I believed that Kinect v1 used Time Of Flight technology or that the the Wii sensor bar transmitted the data to the remote control or the Wii console, I also believed that I could track markers only with the capabilities of Kinect SDK 1.8 or later with OpenNI library.

- One of the biggest problems of this project and to which never I had faced in the career was that **I didn't know if the problem which I wanted to solve had a possible solution**. For example it could happen that I couldn't be able to get the tracking system

to process the images at the required speed or maybe that there was no way to communicate the Kinect camera with the game.

- **I didn't have the resources needed to generate the game** such as 3D models or the audio, so I also had to learn how to use various types of programs such as for instance Blender.

- The project **covered various topics that were being updated throughout the development of the project**, for example during the development of the project were coming to market some virtual reality devices.

- It was a big problem so I got stuck at the beginning of the project trying to gather all the necessary information to define the necessary requirements (I discovered later that for this type of project it was better use agile techniques).

What has to offer?

- **A comprehensive study of the state of the art of the most important actual devices** such as motion controllers or virtual reality devices. To achieve an interesting solution I believed that I had to study the problem from many points of view.

- **A study of problems associated with the use of virtual reality**

- **I demonstrated that it is possible to make use the Kinect V1 for Windows camera to track external objects to the human body** so I was adding extended functionality to the camera.

- **It shows that the chosen system operates smoothly and quickly**. I got 30 updates per second of the position of the controller reaching the maximum frequency because Kinect can get a maximum of 30 images per second. The update rate of the game has also been shown to be optimal getting more than 60 FPS which in terms of Oculus Rift DK1 features is the best possible outcome because the virtual reality device has a refresh rate of 60 Hz.

- **It shows that it is possible to make use older version devices to add new features**.

- **It shows that it is possible to transmit data from a Kinect camera to game engines such as Unity using a UDP port**. In addition also using this method allowed to develop in parallel two applications in different development environments.

- **I show numerous improvements and alternatives**. This has been possible thanks to the study of the state of the art.

- **This work allows to help other people who want to make similar developments**

- **The reading of this paper facilitates the use and installation of OpenCV and OpenNI in Visual Studio 2015**. Since I lost so many time trying to install and configure the project I added an annex with a guide to the installation of the libraries with a small test at the end of each explanation to verify the proper installation.

Goals and achievements at a personal level

- **Be able to adapt to the characteristics of a project as large and with as many unknowns**.

- **Be able to decompose a complex problem in more affordable parts**.

- **I reached most of the initial planning milestones** such as the follow-up reports

- I think I have shown a **great autonomy** which I think it is a key feature for a computer engineer.

During the project development and the study of the different devices and techniques I've been thinking several **improvements and expansions of the final solution**. I think that separate the project in two independent parts (the tracking system and game) has been of great help to develop this kind of project.

Some of the improvements or expansions proposed for the **tracking system** are:

- **Increase the amount of filters** for better recognition of the markers color. For example, if some elements of the background and a marker have a similar color it can lead to failures in the tracking system. The proposed solution is to **implement the background removal filters**. This can be done in several ways, one for example is **trying to give greater weight to objects that move**. To achieve this you can extract the changes between the frames and isolate them from the rest of the data. Another option would be to use the depth map and **delete the data that exceeds a specified level of depth**, or if you would like do it dynamically, calculate the coordinates of the center of the user and delete the data behind him. OpenCV also implements several functions of background removal. I think that adding this type of filter could help **to**

increase the accuracy of the program in a remarkable way, especially when you run the program in certain types of environment.

- You could try to use **another approach to track the object**, for example you can try to track using markers of the same color. **Markers should form a pattern** to be able to **identify** each marker **individually**. This idea was conceived after checking how worked the Kinect v1 system Structured Light (SL). An example of their possible application to the project is to use three eva rubber sheets of the same color on the foam controller. Two would be located in the upper half separated by a space, and the other marker in the lower half, so we could have a pattern. To identify individually each marker the **visibility of the three would be needed**. For example, the upper marker would be the one that had a nearby marker and was the farthest from the other one. **This solution is extensible to more complex patterns** which allow making the pattern recognition even if some markers were not visible by the camera.

- Not only track markers you can also try **track the different parts of the body** and implement it in the game. Although the main objective of this project was to **verify the possibility of using markers to track external objects**, you could also use Kinect tracking system to make the skeleton tracking. This can be achieved for example through the use of the middleware NITE2 along with OpenNI2 and would generate the possibility to **calculate dynamically the minimum and maximum controller position** relative to the position of the player. Another possibility that could lead to its use could be **representing the body of the player in the game**, allowing that he could dodge the projectiles with his own body movement.

- Add **compatibility with standard cameras** eliminating the need to use a Kinect camera, using OpenCV to attempt to get the depth and the relative position of the markers. This could be achieved for example by using a tracking system such as the one used by the PlayStation Eye which is able to get the 3D coordinates of the controller by calculating the size of the marker.

- Add the **possibility to use two Kinect cameras**, placing one in the front of the player and one behind him so the tracking system could **minimize** the cases with **no visible markers**. However this implementation could lead to some problems because the two cameras would interfere each other with the infrared rays but I think it is possible because I saw a video from the company IpiSoft using two Kinect cameras to track a body.

Some of the improvements or expansions for the **game** are:

- **Add filters to stabilize the movement**. Although this case was tried to apply through the development phase I could not achieve a satisfactory result. The problem is that the tracking system can provide incorrect coordinates due failures in color recognition or because of the sun illumination (infrared light from the sun can interfere with the infrared camera making impossible to calculate the depth map). This problem can lead to **sudden jumps in the movement of the light saber**. For instance if the monitoring program detected similar quantities of green color in the top and bottom of a marker but not at half the tracking system will don't know which one is the marker making the central value of the green marker alternate between the top and the bottom, causing

several small jumps in the movement of the light saber. Adding a filter **to detect this type of anomalous behavior** will lead to a more stable movement of the light saber.

- **Make another kind of game or add new functionalities**. Before creating the final game I thought about several approaches. One of the ideas that I considered consisted in the creation of a **linear level** in an industrial complex or spaceship making the player slowly learn the mechanics of the game, something as a tutorial. In this case for instance I was thinking to use a button as the force which would push the objects in front of the player so it could be used also to damage the robot. Although I rejected this idea for the current project, I think it would be interesting to try it, offering the player **several levels** that were **progressively increasing the difficulty** of the game to challenge him.

- **Improve graphics aspects**. Even if I am satisfied with the results considering the time I had I think that it would be interesting to try improve the graphics of the game. For example you could try to **improve the lightsaber shader** to offer a **more realistic** look. You could also try to add a functionality to **modify the color of the blade of the lightsaber**. Another improvement would be the environment by replacing the current 3D models with others **more detailed** could improve the game experience. Changes such as add details to the coliseum like banners, torches, statues, public etc… will help you achieve a better game immersion. You could also texturize 3D models, for example differentiating the different parts of the coliseum.

- **Improve and increase the audio**. Although I think I managed to implement a wide variety of sounds to **avoid the feeling of repetition to the player**, I also think that's possible to **add more types of sounds** that allow increasing the feeling of immersion of the player. For example you could add various sounds of steps to the moving player, add the sound produced by the stone when the player fall or jump over it, the sound of splashing in the water when the player fall etc...

Other improvements and possible similar projects could be:

- **Improve the motion controller**, although I created a controller who works really well with the Kinect camera allowing to get accurate results and its cost is very low, I believe that it is possible to make some improvements to increase the quality of the results of the tracking. For example, there are cases in which markers by lighting issues are not detected correctly by the Kinect camera, especially the upper part of the marker (i.e. top foam controller). I think that **using a sphere instead of a cylindrical surface as a marker** could increase the quality of the tracking. This was raised as an idea in the development of the project and I attempted to carry out, so I bought two polystyrene balls wanting to recover them with eva foam and unite them with a wooden stick. Eventually I discard this kind of controller since the wooden stick could cause damage to the player or the nearby furniture. However I think that it would be interesting to test the effectiveness of the project with modified controllers. Another interesting improvement that can be **implemented** is to add to the controller using Arduino **several inertial sensors** to help calculate the movement of the controller when the camera doesn't have visibility of markers.

- **Make use of the tracking marker system for different applications**, for instance for the education sector. You could try to **create a interactive blackboard** using a marker as a pen and being able to measure the applied pressure using the depth of the marker. Another possibility would be to **make 3D drawings**, making use of an input device such a wireless controller in one hand to handle the color palette or the type of brush and a marker as a brush (a similar application would be Tilt Brush by Google).

- You could try to make an **emulator of other motion control devices** like the HTC Vive, allowing to **substitute the data which they send to their games** with the data of a wireless controller and the data of the motion controlled developed in this project. Obviously the positioning would not be as accurate as the system used by the HTC Vive controllers but it would allow a **cheaper access to virtual reality games**. I think this was interesting so I tried at some point of the development going to the game code "The Lab by Valve" to understand how introduce the values from my tracking system. Due to the complexity of the problem I just abandoned the research. However I believe that it is an interesting approach and that is possible to make it.

- You could try to **use a smartphone as a controller**. At the beginning of the project **was discarded the use of a smartphone** since although he has several inertial sensors (gyroscope accelerometer and magnetometer) the error accumulated by these **not allowed a movement accurate enough**. However this problem **could be solved using the smartphone camera** by placing multiple markers on a fixed point and **using a triangulation technique** similar to the used by the WiiMote. By doing this the triangulation could be used to **correct the error** produced by the inertial sensors, **increasing the accuracy of the** 3D movement tracking. Recognition of markers can be implemented on the mobile itself since the implementation of the OpenCV library in Android devices is possible. Send the data to the computer could be made using Bluetooth for example.

Anexo tutorial configuración TFG para Windows Visual Studio 2015

Instalación Windows Kinect SDK

El proyecto se ha realizado con una cámara Kinect V1 para Windows

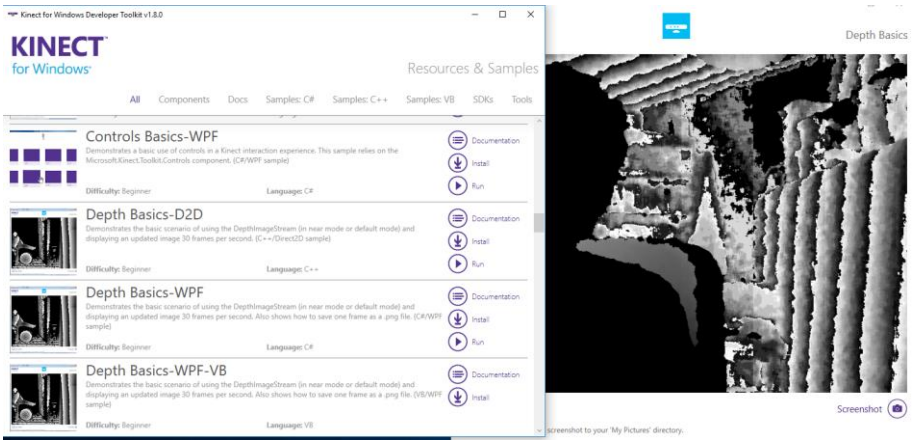
El último SDK destinado para V1 es el SDK 1.8

Lo descargamos desde Microsoft

A continuación nos descargamos Kinect for Windows Developer Toolkit v1.8

Una vez instalado y con la cámara Kinect conectada podremos acceder a los ejemplos y comprobar si funciona nuestra cámara Kinect

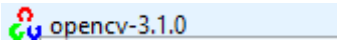
Ejecutamos el ejemplo Depth Basics –WPF, debería poder verse un mapa de profundidad como el que se muestra en la siguiente imagen.



Hay bastantes más ejemplos, sin embargo me he encontrado con el problema de que unos pocos no funcionan.

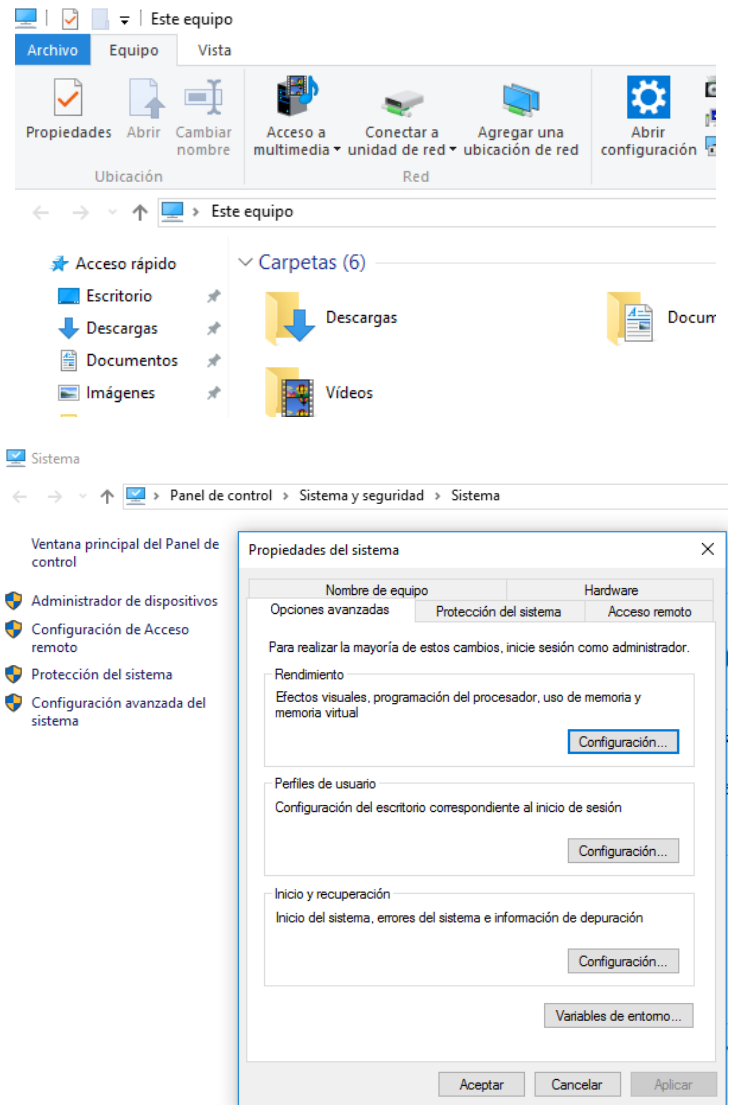
Instalación OpenCV

A continuación descargamos OpenCV, desde la página oficial nos debería dejar descargarnos la más reciente, una vez descargado el exe lo ejecutamos y obtenemos la carpeta OpenCV



Lo más normal es instalarlo directamente en c (c:/opencv), pero también es posible ponerlo en otras rutas (**ojo**, es importante que la ruta **no tenga espacios** ya que si no podrá dar problemas, en mi caso estuve bastante tiempo sin saber dónde estaba el problema).

Ahora tocaría el turno de configurar la variable de entorno de Windows, para ello nos vamos a este equipo→propiedades→configuración avanzada del sistema→Opciones avanzadas→ variables de entorno



Dentro de las variables de entorno si ya hemos instalado openni debería haberse agregado automáticamente la ruta, en mi caso se puede ver en la siguiente imagen.

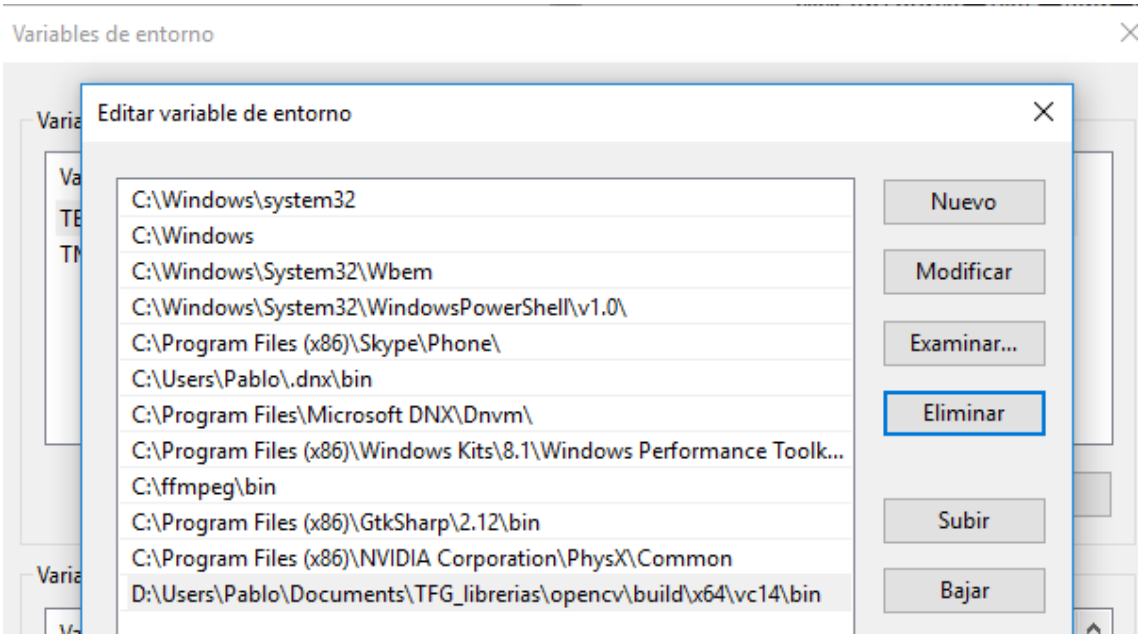
OPENNI2_INCLUDE	D:\Program Files (x86)\OpenNI2\Include\
OPENNI2_INCLUDE64	D:\Program Files\OpenNI2\Include\
OPENNI2_LIB	D:\Program Files (x86)\OpenNI2\Lib\
OPENNI2_LIB64	D:\Program Files\OpenNI2\Lib\
OPENNI2_REDIST	D:\Program Files (x86)\OpenNI2\Redist\
OPENNI2_REDIST64	D:\Program Files\OpenNI2\Redist\

Seleccionamos la variable Path→editar→nuevo

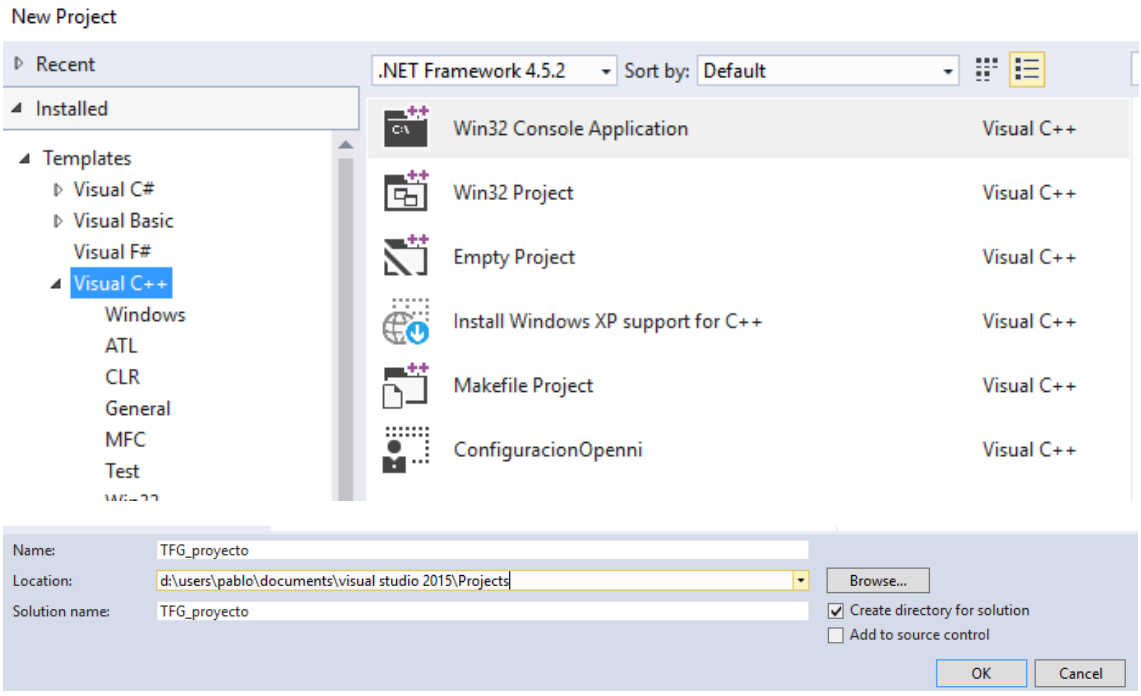
Y agregamos las siguientes rutas (se deberán modificar según el directorio donde lo hayáis instalado)

Ej: D:\Users\Pablo\Documents\TFG_librerias\opencv\build\x64\vc14\bin

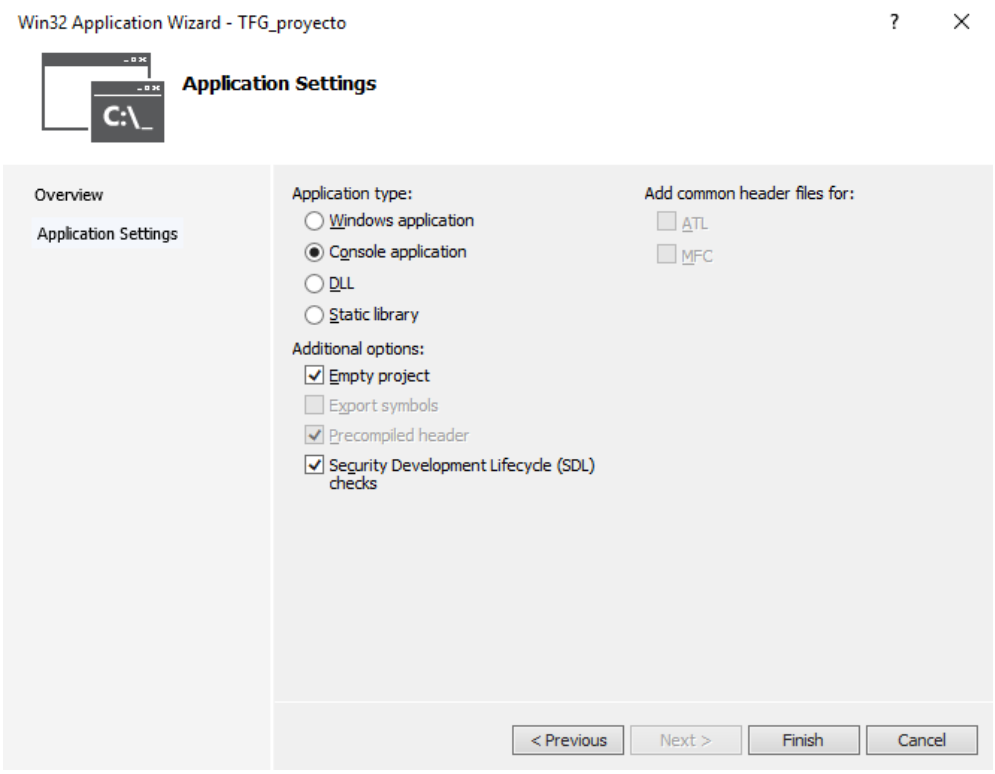
Nota **vc14** es la opción de configuración de **visual 2015**, si tuvierais otro visual studio habría que cambiarlo (v12 hace referencia a visual studio 13)



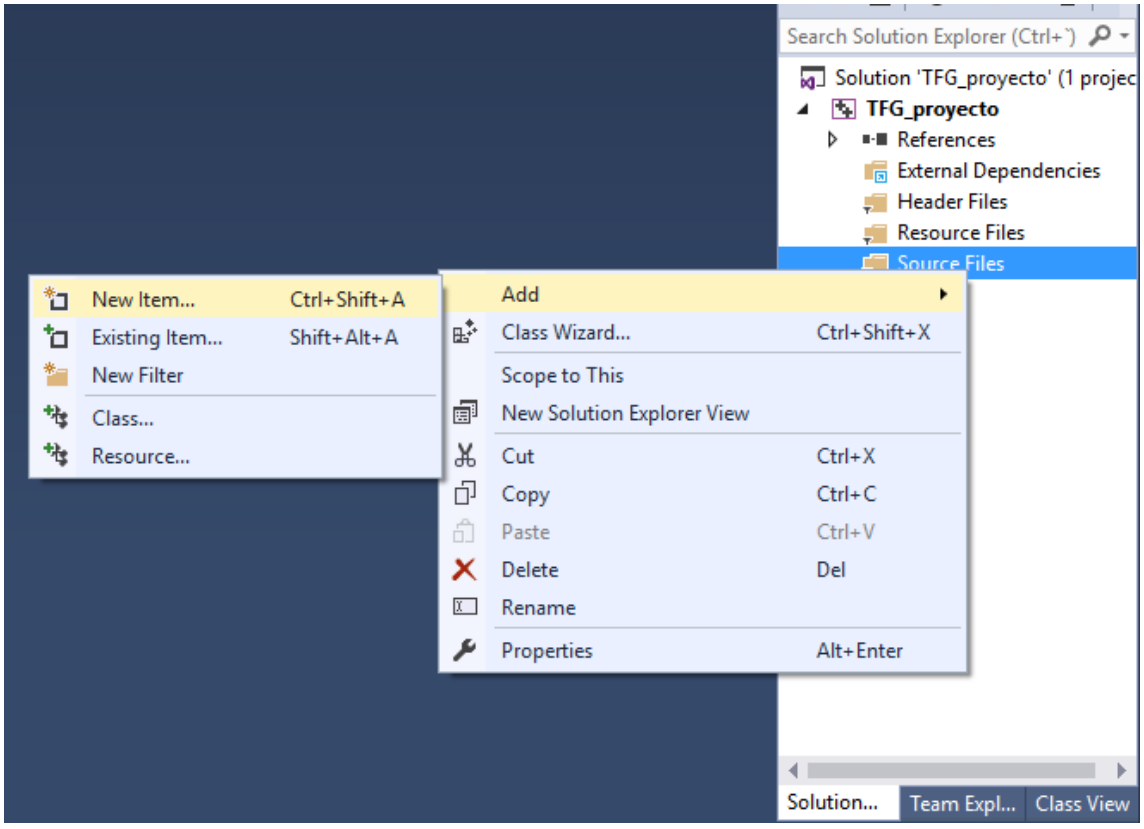
Ya en Visual Studio creamos un nuevo proyecto, Visual c++→ Win32 Console Application (si no aparece esta opción nos aparecerá primero la opción de instalarla).

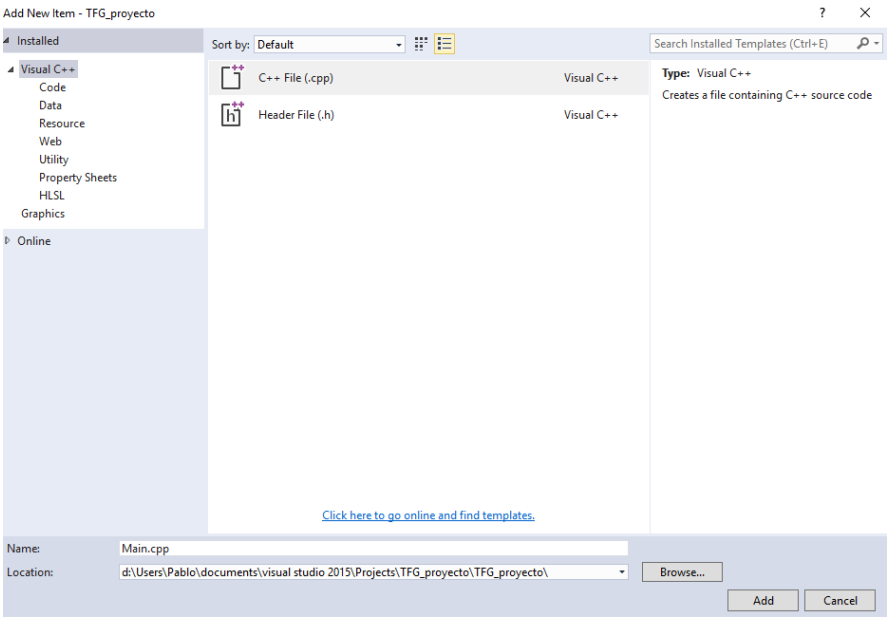


Seleccionamos empty Project y finish

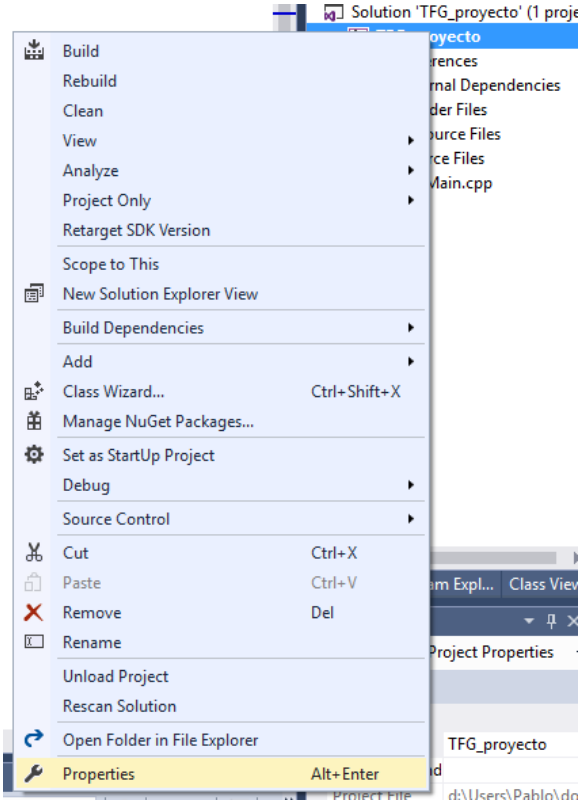


Una vez creado seleccionamos en la ventana superior derecha “Solution Explorer”, segundo click en source→files→add→newItem-→c++ file y le damos el nombre de Main.cpp



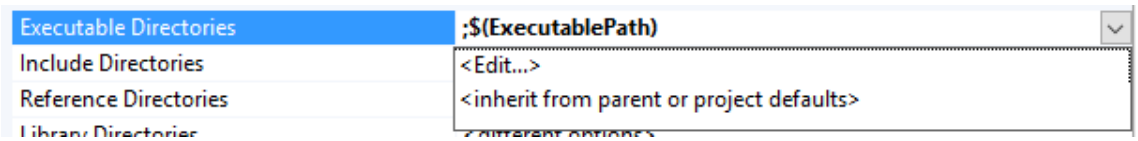


Volvemos acceder a Solution explorer, segundo click en TFG_proyecto (alternativamente podemos acceder desde Project→Properties)

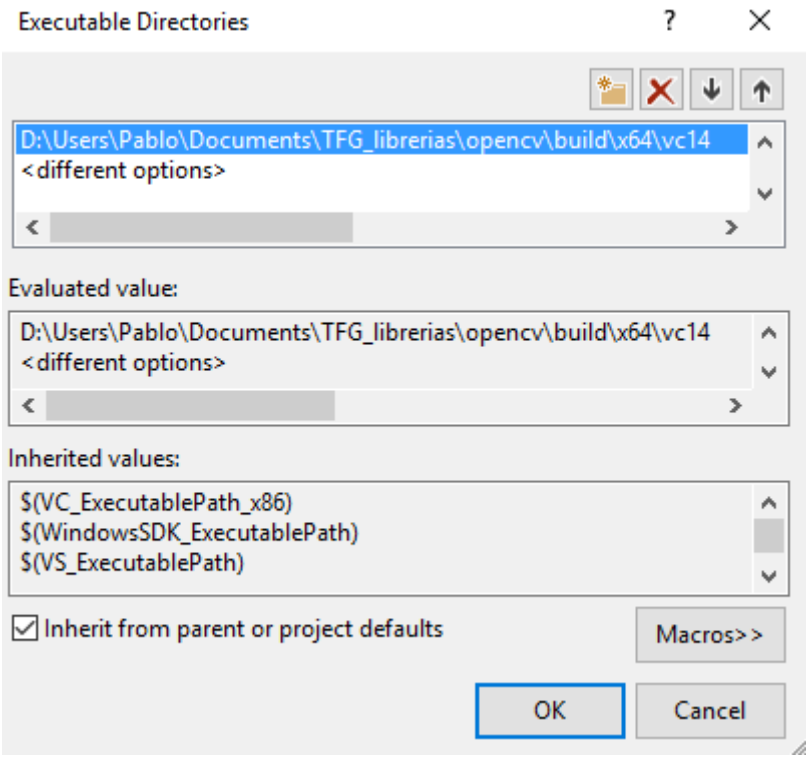


Arriba en configuration seleccionamos configuration: All Configuration y en plataform: all plataforms (realmente opencv solo viene con los archivos x64 así que también podríamos dejarlo en x64)

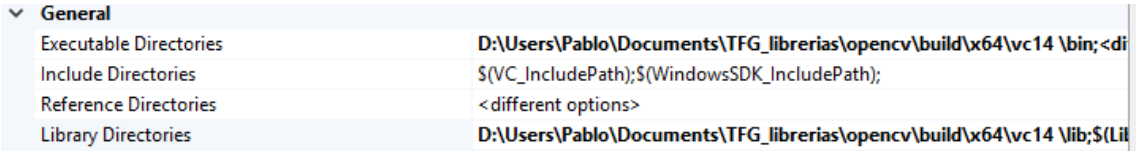
Accedemos a VC++ Directories→executable directories→ edit



Seleccionamos la opción de new Line y pegamos "D:\Users\Pablo\Documents\TFG_librerias\opencv\build\x64\vc14 \bin" (cambiar la ruta por la que tengáis)

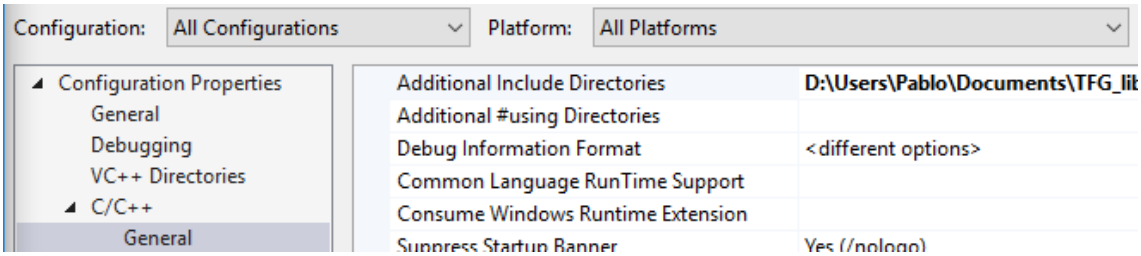


Un poco más abajo en Library Directories agregamos "D:\Users\Pablo\Documents\TFG_librerias\opencv\build\x64\vc14 \lib"



Accedemos a C/C++ →general→Aditional Include Directories agregamos la siguiente ruta

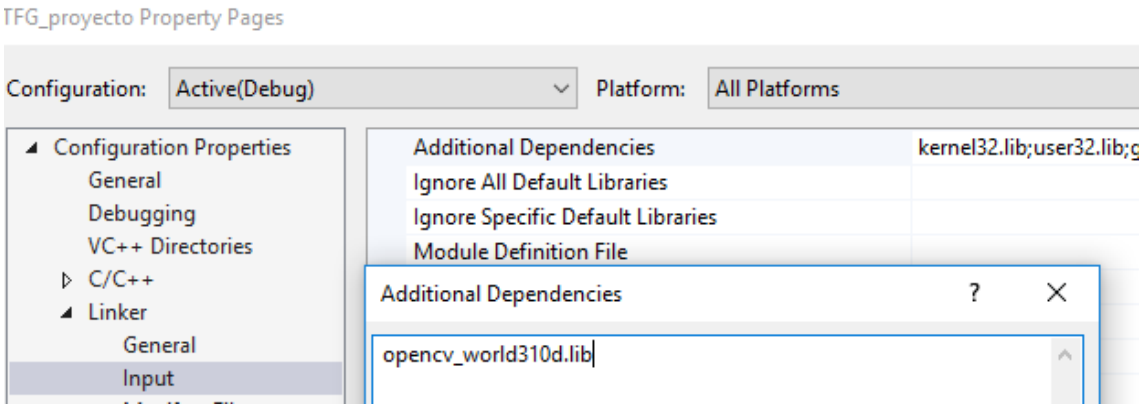
"D:\Users\Pablo\Documents\TFG_librerias\opencv\build\include"



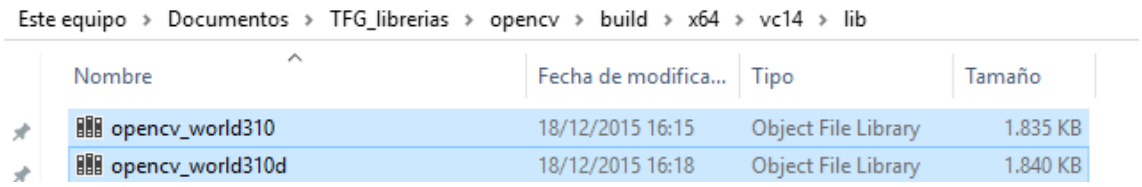
Accedemos a linker→general→ Aditional Library Directories agregamos la carpeta lib

"D:\Users\Pablo\Documents\TFG_librerias\opencv\build\x64\vc14\lib"

Ahora cambiamos configuration: debug, guardamos los datos cuando nos pregunte y accedemos a linker→input→aditional Dependencie y agregamos la librería "opencv_world310d.lib"



Si nos fijamos en la carpeta de las librerías hay 2, la que tiene una d al final hace referencia a la configuración de debug y la que no tiene nada a la de release.



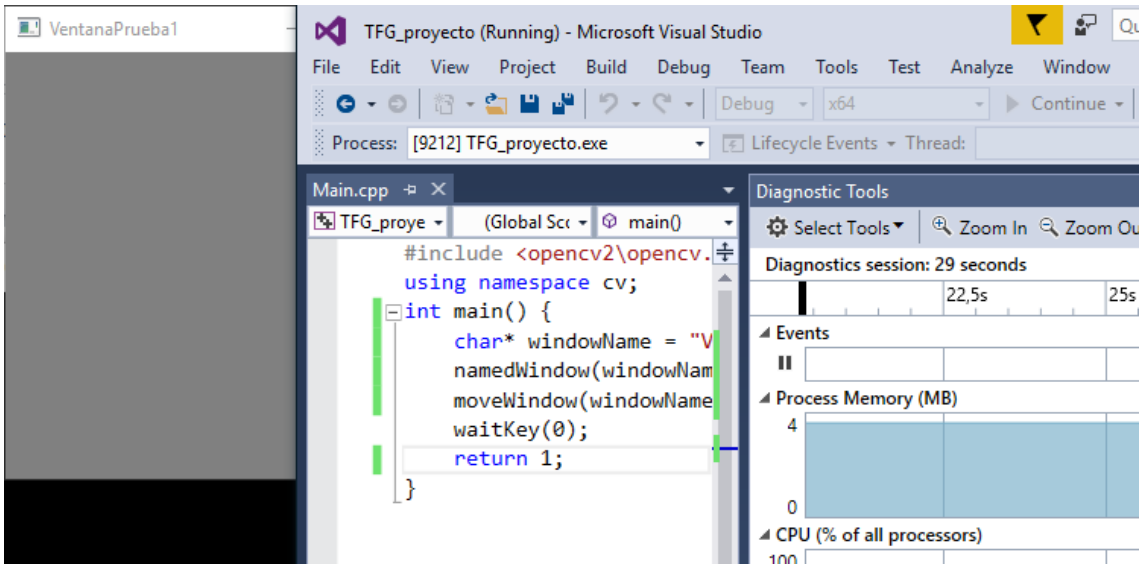
Cambiamos la configuración a Release y agregamos "opencv_world310.lib"

Aplicamos los cambios y aceptamos.

(Si quisiéramos podríamos exportar estas propiedades para utilizarlas en otro proyecto sin tener que repetir todos estos pasos. Para ello tendríamos que usar la opción file→export_template)

Seleccionamos el fichero main.cpp y pegamos el siguiente código, marcamos la opción debug y x64 (como comentábamos antes OpenCV solo viene con los archivos x64 si marcamos la opción x86 nos mostrara un error)

```
#include <opencv2\opencv.hpp>
using namespace cv;
int main() {
	char* windowName = "VentanaPrueba1";
	namedWindow(windowName);
	moveWindow(windowName, 640, 0);
	waitKey(0);
	return 1;
}
```

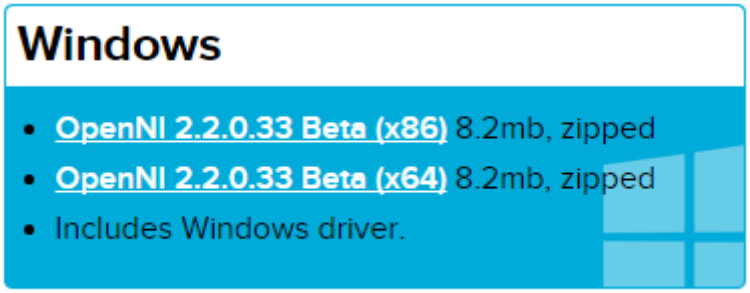


Se nos debería abrir una ventana con el nombre asignado.

En caso de que nos saliera un error del tipo, “falta opencv_world310d.dll” lo más probable es que sea **problema de las variables de entorno**, reinicia el equipo para que se actualicen las variables de entorno y si tras reiniciar sigue dando el mismo problema comprueba que la ruta este bien en path.

Instalación OpenNI2

Nos descargamos de la página oficial la version x64 para Windows



Con la cámara Kinect conectada probaremos si la instalación ha sido correcta ejecutando el programa **NiViewer**, nos debería aparecer un mapa de profundidad al lado del de color.

Con este programa también podremos **grabar archivos de video .ONI** (almacenan los datos de color y de profundidad)

Una vez que hemos comprobado que la instalación ha sido satisfactoria accedemos al proyecto creado en Visual Studio en el tutorial de instalación de OpenCV .

En la ventana solution explorer hacemos click derecho en el proyecto→properties (alternativamente podemos acceder desde Project→Properties)

VC++Directories, nos aseguramos de que la configuración este en “All configurations” y añadimos (la ruta dependerá de donde hayáis instalado OpenNI):

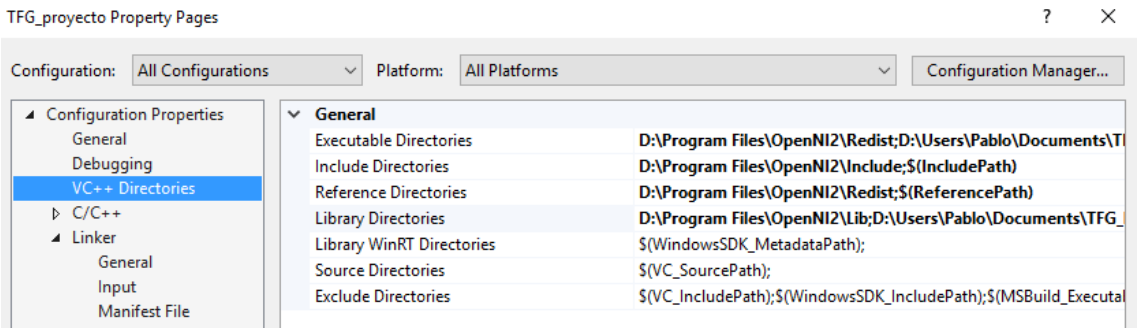
Executables Directories→D:\Program Files\OpenNI2\Redist

Include Directories→D:\Program Files\OpenNI2\Include

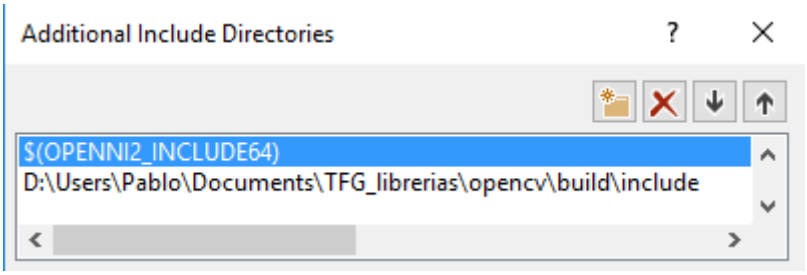
Reference Directories→D:\Program Files\OpenNI2\Redist

Library Directories→D:\Program Files\OpenNI2\Lib

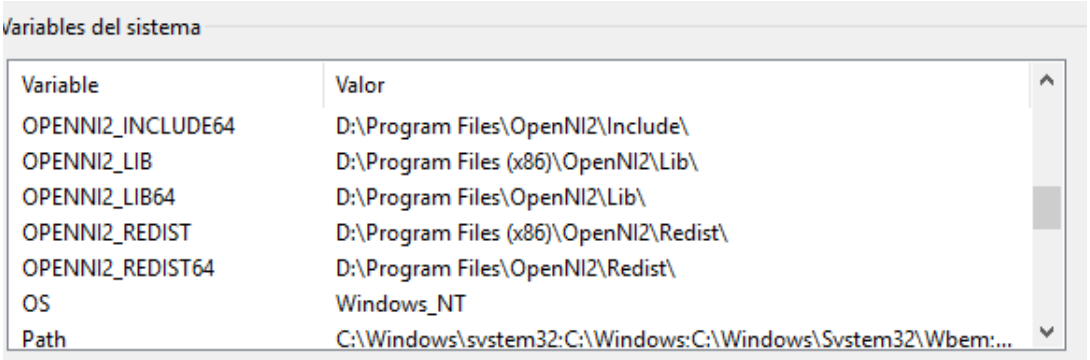
(Nota: Si las rutas llevan espacios entre medias pueden dar problemas)



Ahora accedemos a C/C++→General→Aditional Include Directories y añadimos una nueva línea $(OPENNI2_INCLUDE64)

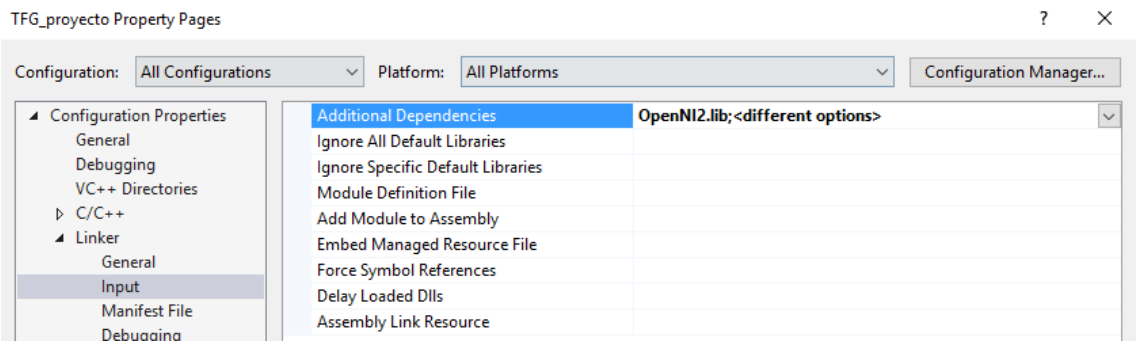


Esto es una variable de entorno que apunta al directorio include de OpenNI de la versión 64 bit, si accedemos a las variables de entorno tal y como se explicaba en el tutorial de instalación de OpenCV podremos ver que al instalar OpenNI se han agregado automáticamente.

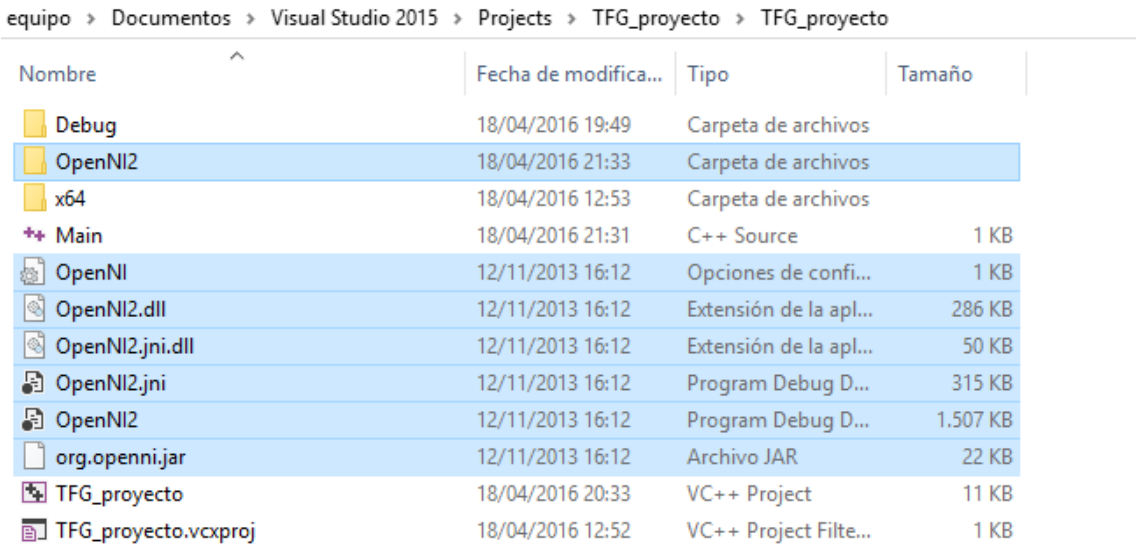


Accedemos ahora a Linker→General→ Aditional Library Directories y agregamos $(OPENNI2_LIB64)

Accedemos a Liker→Input→Aditional Dependencies y agregamos OpenNI2.lib



Aquí hay varias opciones, una es incluir en las variables de entorno en path la dirección a la carpeta redist (en mi caso está en D:\Program Files\OpenNI2\Redist) y reiniciar para que se actualice, o bien copiamos su contenido en la carpeta donde tengamos el ejecutable del proyecto.

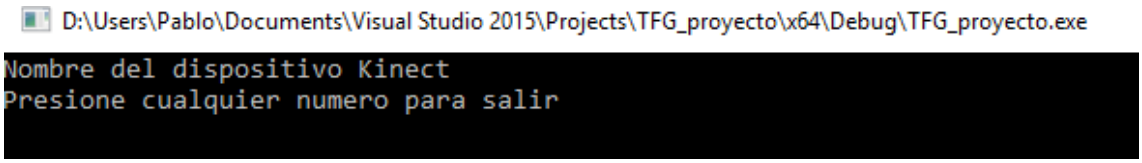


Hacemos una prueba con el siguiente código (deberá estar conectada la cámara Kinect), en caso de que nos muestre un error indicando que falta openni2.obj significa que hemos puesto mal algo en properties→linker→input habrá que asegurarse cambiando la configuración a debug y a release y Plataform x86 y x64 de que no hemos introducido ninguna línea de más en el campo Additional Dependencies.

En caso de que nos dé un error indicando que nos falta Openni2.dll **es un problema del último paso**, en caso de haber elegido la opción de agregar redist a path en las variables de entorno debemos **asegurarnos de haber reiniciado**.

```
#include <OpenNI.h>
#include <iostream>
using namespace openni;
using namespace std;
int main() {
	OpenNI::initialize();
	Device device;
	device.open(ANY_DEVICE);
	printf("Nombre del dispositivo %s \n",device.getDeviceInfo().getName());
	printf("Presione cualquier numero para salir");
	int valorTeclado;
```

```
        cin >> valorTeclado;
        device.close();
        OpenNI::shutdown();
        return 0;
}
```



Bibliografía

[1] Jun-Geun Park. (2013, March).*Quora*. [Online]. Available: https://www.quora.com/Sensors-What-is-the-difference-between-accelerometers-gyroscopes-and-magnetometers

[2] Johnny Chung Lee. (2008).*Hacking the Nintendo Wii Remote*. [Online]. Available: http://www.cs.cmu.edu/~15-821/CDROM/PAPERS/lee2008.pdf

[3] Johnny Chung Lee. (2007, December).*Low-Cost Multi-touch Whiteboard using the Wiimote*. [Online]. Available: https://www.youtube.com/watch?v=5s5EvhHy7eQ

[4] Adrian Biagioli. (2015, October).*Wiimote API for C# and Unity*. [Online]. Available: https://www.youtube.com/watch?v=co7xggFfE94

[5] Eric Ligman. (2010, October 4).*Microsoft Kinect – One month until the gaming & entertainment world changes forever!, blogs.msdn.microsoft.com*. [Online]. Available: https://blogs.msdn.microsoft.com/mssmallbiz/2010/10/04/microsoft-kinect-one-month-until-the-gaming-entertainment-world-changes-forever/

[6] Bill Crounse. (2010, October 24).*Microsoft Health Tech Today—The Future of Natural User Interface (NUI)*. [Online]. Available: https://blogs.msdn.microsoft.com/healthblog/2010/10/24/microsoft-health-tech-todaythe-future-of-natural-user-interface-nui/

[7] Microsoft News Center. (2010, March 31).*PrimeSense Supplies 3-D-Sensing Technology to "Project Natal" for Xbox 360*. [Online]. Available: https://web.archive.org/web/20100620012436/http://www.microsoft.com:80/Presspass/press/2010/mar10/03-31PrimeSensePR.mspx?rss_fdn=Press%20Releases

[8] Kyle Orland. (2010, December).*Kinect 3D Tech Company PrimeSense Releases Open Source PC/Mac Drivers*. [Online]. Available: http://www.gamasutra.com/view/news/31977/Kinect_3D_Tech_Company_PrimeSense_Releases_Open_Source_PCMac_Drivers.php

[9] Richard Mitchell. (2010, October).*PrimeSense releases open source drivers, middleware that work with Kinect, Engadget*. [Online]. Available: https://www.engadget.com/2010/12/10/primesense-releases-open-source-drivers-middleware-for-kinect/

[10] kwc. (2010, December 8).*PrimeSense releases drivers as open source, OpenNI launched, ROS.org*. [Online]. Available: http://www.ros.org/news/2010/12/primesense-releases-drivers-as-open-source-openni-launched.html

[11] Jeffrey Meinser. (2011, February 21).*Kinect for Windows SDK to Arrive Spring 2011. Microsoft Blog*. [Online]. Available: https://blogs.technet.microsoft.com/microsoft_blog/2011/02/21/kinect-for-windows-sdk-to-arrive-spring-2011/

[12] Kyle Orland. (2011, February).*Microsoft Announces Windows Kinect SDK For Spring Release, Gamasutra*. [Online]. Available: http://www.gamasutra.com/view/news/33136/Microsoft_Announces_Windows_Kinect_SDK_For_Spring_Release.php

[13] Mike Isaac. (2011, June 17).*Microsoft Releases Xbox Kinect SDK, Hackers Get to Work, Wired*. [Online]. Available: http://www.wired.com/2011/06/microsoft-kinect-hackers/

[14] Kinect for Windows Team. (2012, January 09).*Starting February 1, 2012: Use the Power of Kinect for Windows to Change the World, Kinect for Windows Product Blog*. [Online]. Available: https://blogs.msdn.microsoft.com/kinectforwindows/2012/01/09/starting-february-1-2012-use-the-power-of-kinect-for-windows-to-change-the-world/

[15] Kinect for Windows Team. (2012, January 20).*Near Mode: What it is (and isn't), Kinect for Windows Product Blog*. [Online]. Available: https://blogs.msdn.microsoft.com/kinectforwindows/2012/01/20/near-mode-what-it-is-and-isnt/

[16] Kinect for Windows Team. (2012, January 31).*Kinect for Windows is now Available!, Kinect for Windows Product Blog*. [Online]. Available: https://blogs.msdn.microsoft.com/kinectforwindows/2012/01/31/kinect-for-windows-is-now-available/

[17] Microsoft. (2012, May 1).*Kinect for Windows SDK Beta 2, Microsft Download Center*. [Online]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=27876

[18] Kinect for Windows Product Team. (2013, September 17).*Updated SDK, with HTML5, Kinect Fusion improvements, and more, Kinect for Windows Product Blog*. [Online]. Available: https://blogs.msdn.microsoft.com/kinectforwindows/2013/09/17/updated-sdk-with-html5-kinect-fusion-improvements-and-more/

[19] Leigh Alexander. (2011, March 9).*Microsoft: Kinect Hits 10 Million Units, 10 Million Games, Gamasutra*. [Online]. Available: http://www.gamasutra.com/view/news/123831/Microsoft_Kinect_Hits_10_Million_Units_

[20] Kinect for Windows Team. (2014, July 15).*The Kinect for Windows v2 sensor and free SDK 2.0 public preview are here, Kinect for Windows Product Blog*. [Online]. Available: https://blogs.msdn.microsoft.com/kinectforwindows/2014/07/15/the-kinect-for-windows-v2-sensor-and-free-sdk-2-0-public-preview-are-here/

[21] Microsoft Corporate Blogs. (2014, October 22).*Microsoft releases Kinect SDK 2.0 and new adapter kit, Microsoft Blog*. [Online]. Available: http://blogs.microsoft.com/blog/2014/10/22/microsoft-releases-kinect-sdk-2-0-new-adapter-kit/#sm.0000em1w2q3nfcvbx7f28e0avv5zx

[22] Kinect for Windows Team. (2015, April 2).*Microsoft to consolidate the Kinect for Windows experience around a single sensor, Kinect for Windows Product Blog*. [Online]. Available: https://blogs.msdn.microsoft.com/kinectforwindows/2015/04/02/microsoft-to-consolidate-the-kinect-for-windows-experience-around-a-single-sensor/

[23] Microsoft. *Kinect for Windows Sensor Components and Specifications, Microsoft Developer*. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131033.aspx

[24] Microsoft. *Color Stream, Microsoft Developer*. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131027.aspx

[25] Microsoft. *Depth Stream, Microsoft Developer*. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131028.aspx

[26] Microsoft. *Features SDK 2.0, Microsoft Developer*. [Online]. Available: https://msdn.microsoft.com/library/dn782025.aspx

[27] Microsoft. *Kinect V2 Hardware, Windows Dev Center*. [Online]. Available: https://developer.microsoft.com/en-us/windows/kinect/hardware

[28] James Ashley. (2014, March 5).*QUICK REFERENCE: KINECT 1 VS KINECT 2*. [Online]. Available: http://www.imaginativeuniversal.com/blog/post/2014/03/05/Quick-Reference-Kinect-1-vs-Kinect-2.aspx

[29] Alejandro Murillo. *Diferencias entre Kinect Xbox 360 y Kinect for Windows, Kinect for Developers*. [Online]. Available: http://www.kinectfordevelopers.com/es/2012/03/01/diferencias-entre-kinect-xbox-360-y-kinect-for-windows/

[30] Jeff Kramer, Nicolas Burrus, Florian Echtler, Daniel Herrera C., and Matt Parker, *Hacking the Kinect*, 1st ed.: Apress, 2012.

[31] Hamed Sarbolandi, Damien Lefloch, and Andreas Kolb, "Kinect range sensing: Structured-light versus Time-of-Flight Kinect," *Computer vision and image understanding*, vol. 139, pp. 1-20, May 2015.

[32] CuriousInventor. (2013, February 16).*How the Kinect Depth Sensor Works in 2 Minutes*. [Online]. Available: https://www.youtube.com/watch?v=uq9SEJxZiUg

[33] Trevor Taylor. (2011, November 29).*Kinect for Robotics, Microsoft Robotics Blog*. [Online]. Available: https://blogs.msdn.microsoft.com/msroboticsstudio/2011/11/29/kinect-for-robotics/

[34] Daniel Lau. (2013, November 27).*The Science Behind Kinects or Kinect 1.0 versus 2.0, Gamasutra*. [Online]. Available: http://www.gamasutra.com/blogs/DanielLau/20131127/205820/The_Science_Behind_Kinects_or_Kinect_10_versus_20.php

[35] Nassir Navab Victor Castaneda. (2011, June 1).*Time-of-Flight and Kinect Imaging*. [Online]. Available: http://campar.in.tum.de/twiki/pub/Chair/TeachingSs11Kinect/2011-DSensors_LabCourse_Kinect.pdf

[36] R. A., Jidong Huang, and M. Yeh El-laithy, "Study on the use of microsoft kinect for robotics applications," in *Proceedings of the 2012 IEEE/ION Position, Location and Navigation Symposium*, 2012, pp. 1280 - 1288.

[37] Microsoft. *Skeleton Tracking With Multiple Kinect Sensors, Microsoft Developer*. [Online]. Available: https://msdn.microsoft.com/es-es/enus/library/dn188677.aspx?f=255&MSPPError=-2147217396

[38] Bradley Austin, Philip Rosedale, Karen Bryla, and Philips Alexander Benton Davis, *Oculus rift in action*, 1st ed.: Shelter Island, 2015.

[39] Robert Purchese. (2016, July 11).*Happy Go Luckey: Meet the 20-year-old creator of Oculus Rift, Eurogamer*. [Online]. Available: http://www.eurogamer.net/articles/2013-07-11-happy-go-luckey-meet-the-20-year-old-creator-of-oculus-rift

[40] *Oculus VR: The Story So Far, dSky*. [Online]. Available: http://dsky9.com/rift/oculus-rift-the-story-so-far/

[41] Oculus. (2012, August 14).*Former Gaikai and Scaleform Officer, Brendan Iribe, Joins Oculus VR™ as CEO*. [Online]. Available: https://www1.oculus.com/press/former-gaikai-and-scaleform-officer-brendan-iribe-joins-oculus-as-ceo/

[42] Oculus. *KickStarter Oculus Rift, KickStarter*. [Online]. Available: https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game

[43] Oculus. (2012, November 28).*Update on Developer Kit Technology, Shipping Details, Oculus Blog*. [Online]. Available: https://www.oculus.com/en-us/blog/update-on-developer-kit-technology-shipping-

[44] Stephen Totilo. (2013, Agust 7).*John Carmack Has New 'Full-Time' VR Job, But Is Not Quite Gone From id, Kotaku*. [Online]. Available: http://kotaku.com/john-carmack-has-a-new-job-but-still-involved-with-1053820115

[45] Evan Narcisse. (2013, November 22).*Doom Co-Creator John Carmack Leaves Id Software, Kotaku*. [Online]. Available: http://kotaku.com/doom-co-creator-john-carmack-leaves-id-software-1469878905

[46] Oculus. (2014, March 19).*Announcing the Oculus Rift Development Kit 2 (DK2), Oculus Blog*. [Online]. Available: https://www.oculus.com/en-us/blog/announcing-the-oculus-rift-development-kit-2-dk2/

[47] Stuart Dredge. (2014, July 22).*Facebook closes its $2bn Oculus Rift acquisition. What next?, theguardian*. [Online]. Available: https://www.theguardian.com/technology/2014/jul/22/facebook-oculus-rift-acquisition-virtual-reality

[48] Oculus. (2014, September 20).*Oculus Connect 2014, Oculus Blog*. [Online]. Available: https://www.oculus.com/en-us/blog/oculus-connect-2014/

[49] Oculus. (2016, January 4).*Oculus Rift Pre-Orders to Open on January 6, Oculus Blog*. [Online]. Available: https://www.oculus.com/en-us/blog/oculus-rift-pre-orders-to-open-on-jan-6/

[50] Ben Lang. (2016, April 24).*Some Oculus Rift Orders Shipping Significantly Ahead of Delay Estimates, RoadToVR*. [Online]. Available: http://www.roadtovr.com/some-oculus-rift-orders-shipping-significantly-ahead-of-delay-estimates/

[51] Michael Abrash. (2013, July 26).*Down the VR rabbit hole: Fixing judder, Blog Michael Abrash*. [Online]. Available: http://blogs.valvesoftware.com/abrash/down-the-vr-rabbit-hole-fixing-judder/

[52] eVRydayVR. (2014, January 8).*Discussion: New features of the Oculus Rift Crystal Cove prototype from CES*. [Online]. Available: https://www.youtube.com/watch?v=HoLHHUdi_LE&feature=youtu.be&t=6m17s

[53] BlurBusters. *Motion Blur Demo*. [Online]. Available: http://www.testufo.com/#test=eyetracking&pattern=stars

[54] Xin Reality VR&AR wiki. *Oculus Touch*. [Online]. Available: http://xinreality.com/wiki/Oculus_Touch

[55] Michael McWhertor. (2014, March 18).*Sony announces Project Morpheus, a virtual reality headset coming to PlayStation 4, Polygon*. [Online]. Available: http://www.polygon.com/2014/3/18/5524058/playstation-vr-ps4-virtual-reality

[56] Jamie Feltham. *Project Morpheus Started as 'grassroots activity' Following PS Move's Release*. [Online]. Available: http://www.vrfocus.com/2014/12/project-morpheus-started-grassroots-activity-following-ps-moves-release/

[57] Sony. (2015, Oct 8).*Sony Acquires Belgian Innovator of Range Image Sensor Technology, Softkinetic Systems S.A., in its Push Toward Next-Generation Range Image Sensors and Solutions*. [Online]. Available: http://www.sony.net/SonyInfo/News/Press/201510/15-083E/

[58] Jeff Grubb. (2016, March 17).*Sony will reject PlayStation VR games that aren't at least 60 frames per second, VB*. [Online]. Available: http://venturebeat.com/2016/03/17/sony-will-reject-playstation-vr-games-that-arent-at-least-60-frames-per-second/

[59] Ben Lang. (2014, September 24).*Oculus Shares 5 Key Ingredients for Presence in Virtual Reality, RoadToVR*. [Online]. Available: http://www.roadtovr.com/oculus-shares-5-key-ingredients-for-presence-in-virtual-reality/

[60] Paul James. (2015, March 5).*Valve Reveals Timeline of Vive Prototypes, We Chart it For You, RoadToVR*. [Online]. Available: http://www.roadtovr.com/valve-reveals-timeline-of-vive-prototypes-we-chart-it-for-you/

[61] (2013, March 27).*What are AprilTags?, mit.edu*. [Online]. Available: http://people.csail.mit.edu/kaess/apriltags/

[62] Edwin Olson. *AprilTag: A robust and flexible visual fiducial system*. [Online]. Available: https://april.eecs.umich.edu/pdfs/olson2011tags.pdf

[63] Oculus. (2013, March 18).*Team Fortress 2 in the Oculus Rift, Oculus Blog*. [Online]. Available: https://www.oculus.com/en-us/blog/team-fortress-2-in-the-oculus-rift/

[64] rvdm88. (2015, August 24).*HTC Vive Lighthouse Chaperone tracking system Explained*. [Online]. Available: https://www.youtube.com/watch?v=J54dotTt7k0

[65] XinRealityVR&ARWiki. *Lighthouse*. [Online]. Available: http://xinreality.com/wiki/Lighthouse

[66] Sean Buckley. (2015, May 19).*This Is How Valve's Amazing Lighthouse Tracking Technology Works, Gizmodo*. [Online]. Available: http://gizmodo.com/this-is-how-valve-s-amazing-lighthouse-tracking-technol-1705356768

[67] Paul James. (2014, June 26).*News Bits: Impressive VR Lightsaber Video Uses the Oculus Rift and a Serious Mo-Cap Rig, RoadToVR*. [Online]. Available: http://www.roadtovr.com/news-bits-impressive-vr-lightsaber-video-uses-oculus-rift-serious-mo-cap-rig/

[68] heise online. (2014, November 5).*Jedi Cave: Star Wars im selbstgebauten Holodeck*. [Online]. Available: https://www.youtube.com/watch?v=5W1wyhLcpso

[69] OptiTrack. *OptiTrack camera Flex 13*. [Online]. Available: http://www.optitrack.com/products/flex-13/

[70] Antonio Mauro Galiano. (2013, December 2013).*Robust Colored Objects And Marker Tracking (kinect opencv)*. [Online]. Available: https://www.youtube.com/watch?v=8Lc7yk7KGDo

[71] Ian Sommerville, *Ingeniería del software*, 7th ed.: Pearson Addison Wesley, 2005.

[72] UC3M. (2010, Junio 17).*NORMATIVA SOBRE LA ORGANIZACIÓN Y EVALUACIÓN DE LA ASIGNATURA "TRABAJO FIN DE GRADO"*. [Online]. Available: http://portal.uc3m.es/portal/page/portal/organizacion/secret_general/normativa/estudiantes/estudios_grado/NormativaTrabajoFindeGrad_definitiva.pdf

[73] Apache Software Foundation. (2004, January).*Apache License Version 2.0*. [Online]. Available: http://www.apache.org/licenses/LICENSE-2.0

[74] Kyle Hounslow. (2015).*ObjectTrackingTutorial.cpp*. [Online]. Available: https://raw.githubusercontent.com/kylehounslow/opencv-tuts/master/auto-colour-filter/AutoColourFilter.cpp

[75] Hasan Azizul Haque Avi. (2014, February 1).*OpenCV Object Tracking using CamShift algorithm and Unity3d Mashup*. [Online]. Available: https://www.youtube.com/watch?v=aE4_VtUDORw

[76] Tom Vian. (2010).*As3sxfr*. [Online]. Available: http://www.superflashbros.net/as3sfxr/

[77] Madison Pike. *Madison Pike LLC Asset License*. [Online]. Available: http://pastebin.com/Jc4YAeGt

[78] AEVI Asociación española de videojuegos. (2016, Marzo 30).*El consumo global de videojuegos en España superó los 1.000 millones de euros en 2015*. [Online]. Available: http://www.aevi.org.es/consumo-global-videojuegos-espana-supero-los-1-000-millones-euros-2015/

[79] Marc Hoag. (2015, November 4).*Google vs. Tesla: Two different philosophies on self-driving cars*. [Online]. Available: https://innovately.wordpress.com/2015/11/04/google-vs-tesla-two-different-philosophies-on-self-driving-cars/

[80] Ashlee Vance. (2015, December 16).*The First Person to Hack the iPhone Built a Self-Driving Car. In His Garage, Bloomberg Buinessweek*. [Online]. Available: http://www.bloomberg.com/features/2015-george-hotz-self-driving-car/

[81] Michael Zelenko. (2016, June 6).*On the road with George Hotz's $1,000 self-driving car kit, The Verge*. [Online]. Available: http://www.theverge.com/2016/6/6/11866868/comma-ai-george-hotz-interview-self-driving-cars

[82] Sarah E. Needleman. (2016, June 13).*Microsoft Unveils New Virtual-Reality Gaming Console, The Wall Street Journal*. [Online]. Available: http://www.wsj.com/articles/microsoft-unveils-new-virtual-reality-gaming-console-1465847139

[83] Deloitte Global. *Virtual reality (VR): a billion dollar niche*. [Online]. Available: http://www2.deloitte.com/global/en/pages/technology-media-and-telecommunications/articles/tmt-pred16-media-virtual-reality-billion-dollar-niche.html

[84] Digi-Capital. (2016, January).*Augmented/Virtual Reality revenue forecast revised to hit $120 billion by 2020*. [Online]. Available: http://www.digi-capital.com/news/2016/01/augmentedvirtual-reality-revenue-

forecast-revised-to-hit-120-billion-by-2020/#.V2bahriLTct

[85] Amazon. *Kinect for Windows v1, Amazon*. [Online]. Available: https://www.amazon.com/Microsoft-L6M-00001-Kinect-for-Windows/dp/B006UIS53K

[86] DreamSpark. *DreamSpark*. [Online]. Available: https://e5.onthehub.com/WebStore/ProductsByMajorVersionList.aspx?ws=8738733a-6d9b-e011-969d-0030487d8897&vsro=8&JSEnabled=1&pc=0dafd5cd-4c09-e011-bed1-0030487d8897

[87] VBandi. (2013, March 25).*Kinect Interactions with WPF - Part I: Getting Started*. [Online]. Available: http://dotneteers.net/blogs/vbandi/archive/2013/03/25/kinect-interactions-with-wpf-part-i-getting-started.aspx

[88] Pankaj Deharia Ignatiuz. (2013, December 2013).*Kinect status and setup the Kinect for interaction, Code Project*. [Online]. Available: http://www.codeproject.com/Tips/701338/Kinect-status-and-setup-the-Kinect-for-interaction

[89] Microsoft. (2013, September 13).*Kinect for Windows Developer Toolkit v1.8*. [Online]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=40276

[90] Falahati Soroush, *OpenNI cookbook*, 1st ed.: Packt Pub, 2013.

[91] Kyle Hounslow. (2015, September 19).*OpenCV Tutorial: Adding an Automatic Colour Filter for Object Tracking*. [Online]. Available: https://www.youtube.com/watch?v=s5ROKdRUkuI

[92] Tienda Malabares. (2011, August 10).*COMO MALABAREAR PELOTAS - CASCADA 3 PELOTAS - TIENDA MALABARES.COM*. [Online]. Available: https://www.youtube.com/watch?v=bfO5updXN6s

[93] CMake. *CMake*. [Online]. Available: https://cmake.org/

[94] OpenCV. *Using Kinect and other OpenNI compatible depth sensors*. [Online]. Available: http://docs.opencv.org/3.0-rc1/d7/df5/tutorial_ug_highgui.html

[95] Open Broadcaster Software. *OBS*. [Online]. Available: https://obsproject.com/

[96] Oculus. *Introduction to Best Practices*. [Online]. Available: https://developer.oculus.com/documentation/intro-vr/latest/concepts/bp_intro/

[97] milanith. (2016, March 9).*Lightsaber shader - Unity 5*. [Online]. Available: https://www.youtube.com/watch?v=26075PWXrE0

[98] CG Geek. (2015, September 19).*Blender Beginner Tutorial: Create a Lightsaber - 1 of 2*. [Online]. Available: https://www.youtube.com/watch?v=YbsYDOiph7U

[99] Robert Langanière, *OpenCV Computer Vision Application Programming Cookbook Second Edition*, 2nd ed.: Packt Publishing, 2014.

[100] Slavik D Tabakov, "INTRODUCTION TO VISION, COLOUR MODELS AND IMAGE," *MEDICAL PHYSICS INTERNATIONAL*, vol. 1, no. 1, 2013.

[101] Joseph Howse, *OpenCV 3 Blueprints*.: Packt Publishing, 2015.

[102] OpenCV. *Structural Analysis and Shape Descriptors*. [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html

[103] OpenCV. *Basic Drawing*. [Online]. Available: http://docs.opencv.org/2.4/doc/tutorials/core/basic_geometric_drawing/basic_geometric_drawing.html

[104] Microsoft. *Sendto function, Windows Dev Center*. [Online]. Available: https://msdn.microsoft.com/en-

us/library/windows/desktop/ms740148%28v=vs.85%29.aspx?f=255&MSPPError=-2147217396

[105] Speed guide. *Port 5005 Details*. [Online]. Available: http://www.speedguide.net/port.php?port=5005

[106] Tech-FAQ. *127.0.0.1 – What Are its Uses and Why is it Important?* [Online]. Available: http://www.tech-faq.com/127-0-0-1.html

[107] iPiSoft. (2011, November 18).*iPi Desktop Motion Capture with 2 Kinect - demo 1*. [Online]. Available: https://www.youtube.com/watch?v=msRtIZX529Q